Adabas

Utilities Manual, Volume 1

Manual Order Number: ADA741-080IBB

This document applies to Adabas Version 7.4 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Readers' comments are welcomed. Comments may be addressed to the Documentation Department at the address on the back cover or to the following e-mail address:

Documentation@softwareag.com

TABLE OF CONTENTS

GENERAL INFORMATION 1

About This Manual 1
Utility Control Statements 2
Control Statement Syntax 2
Control Statement Rules 6
Parameter Values 6

1. ADAACK : CHECK ADDRESS CONVERTER 11

Functional Overview 11
ACCHECK :
Check Address Converter Against Data Storage 12
Optional Parameters 12
Examples 13
JCL/JCS Requirements and Examples 14
BS2000 14
z/OS or OS/390 16
z/VM or VM/ESA 17
VSE/ESA 18

2. ADACDC : CHANGED-DATA CAPTURE 19

Functional Overview 19
Phases of Operation and Resulting Files 19
Primary Input Data 22
Primary Output Data 22
Transaction File 23
Running the Utility 25
Optional Parameters 25
Operating System Considerations 27
z/OS or OS/390 27
VSE/ESA 28

BS2000 28
The ADACDC User Exit 29
Installing the Exit 29
User Exit Interface 29
User Exit Calls 31
Examples 33
JCL/JCS Requirements and Examples 34
BS2000 34
z/OS or OS/390 36
z/VM or VM/ESA 38
VSE/ESA 40

3. ADACMP : COMPRESS–DECOMPRESS 43

Functional Overview 43
Overview of the COMPRESS Function 43
Overview of the DECOMPRESS Function 44
Input Data Requirements 45
Input Data Structure 45
Multiple-Value Field Count 45
Periodic Group Count 47
Variable-Length Field Size 50
Processing 52
Data Verification 52
Data Compression 52
COMPRESS Function Output 54
Compressed Data Records 54
Rejected Data Records 54
ADACMP Report 55
DECOMPRESS Function Output 57
Rejected Data Records 58
Restart Considerations 59
User Exit 6 59

COMPRESS : Create an Adabas File 60
Optional Parameters and Subparameters 61
Essential Data Definition Syntax 66
Optional Field Definition Statements 83
ADACMP COMPRESS Examples 101
DECOMPRESS : Decompress File(s) 103
Optional Parameters and Subparameters 103
Decompressing Multiclient Files 107
ADACMP DECOMPRESS Examples 108
JCL/JCS Requirements and Examples 108
User Exits with ADACMP 108
BS2000 110
OS/390 or z/OS 113
VM/ESA or z/VM 116
VSE/ESA 117

4. ADACNV : DATABASE CONVERSION 121

Functional Overview 121
Database Status 121
Procedure 122
CONVERT : Convert Database to Higher Version 123
Optional Parameters 123
Conversion Considerations 124
Example 125
REVERT : Revert Database to Lower Version 126
Essential Parameter and Subparameter 126
Optional Parameter 126
Reversion Considerations 127
Example 128
JCL/JCS Requirements and Examples 129
BS2000 129
OS/390 or z/OS 132
VM/ESA or z/MV 133
VSE/ESA 134

5. ADADBS : DATABASE SERVICES 137

Functional Overview 137
Syntax Checking with the TEST Parameter 138
ADD : Add Dataset 139
Associator or Data Storage Dataset 139
Essential Parameter and Subparameter 140
Optional Parameters 140
Examples 140
ALLOCATE : Allocate File Extent 141
Essential Parameters 141
Optional Parameters 141
Example 142
CHANGE : Change Standard Length of a Field 143
Essential Parameters 143
Optional Parameters 144
Example 145
CVOLSER : Print Adabas Extents on Given Volume 145
Essential Parameter 145
Optional Parameters 145
Example 146
DEALLOCATE : Deallocate File Extent 146
Essential Parameters 146
Optional Parameters 147
Example 147
DECREASE : Decrease Associator/Data Storage 148
Essential Parameter 148
Optional Parameters 148
Example 149
Procedure 149
DELCP : Delete Checkpoint Records 149
Essential Parameter 149
Optional Parameters 150
Example 150

DELETE : Delete File 151
Essential Parameter 151
Optional Parameters 151
Examples 152
DSREUSE : Reuse Data Storage Blocks 153
Essential Parameters 153
Optional Parameters 154
Example 154
ENCODEF : Change File Encoding 155
Essential Parameter 155
Optional Parameters 155
Example 156
INCREASE : Increase Associator/Data Storage 156
Essential Parameter 157
Optional Parameters 157
Example 157
General Procedure 157
Operating-System-Specific Procedures 158
ISNREUSE : Reuse ISNs 164
Essential Parameters 164
Optional Parameters 164
Example 165
MODFCB : Modify File Parameters 165
Essential Parameter 166
Optional Parameters 166
Example 167
NEWFIELD : Add New Field 168
Essential Parameter 168
Optional Parameters 169
Example 170
ONLINVERT : Start Online Invert 171
Essential Parameters 171
Optional Parameters 172
Example 173

ONLREORFASSO :
Start Online Reorder Associator for Files 173
Essential Parameters 174
Optional Parameters 174
Example 175
ONLREORFDATA : Start Online Reorder Data for Files 175
Essential Parameters 176
Optional Parameters 176
Example 177
ONLREORFILE :
Start Online Reorder Associator and Data for Files 178
Essential Parameters 178
Optional Parameters 179
Example 180
OPERCOM : Adabas Operator Commands 181
Using OPERCOM Commands in Cluster Environments 181
Optional Parameters 182
Operator Commands 183
PRIORITY : Change User Priority 198
Essential Parameter 198
Optional Parameters 198
Example 199
RECOVER : Recover Space 199
Optional Parameters 199
REFRESH : Set File to Empty Status 200
Essential Parameter 200
Optional Parameters 200
Example 201
REFRESHSTATS : Refresh Statistical Values 201
Optional Parameters 201
Example 203
RELEASE : Release Descriptor 203
Essential Parameters 203
Optional Parameters 204
Example 204

RENAME : Rename File/Database 205
Essential Parameter 205
Optional Parameters 205
Examples 206
RENUMBER : Change File Number 206
Essential Parameter 206
Optional Parameter 206
Example 207
RESETDIB : Reset Entries in Active Utility List 207
Essential Parameters 208
Optional Parameters 208
Examples 208
TRANSACTIONS : Suspend and Resume Transactions 209
Essential Parameters 210
Optional Parameters 211
Example 211
UNCOUPLE : Uncouple Files 212
Essential Parameter 212
Optional Parameters 212
Example 213
JCL/JCS Requirements and Examples 213
Collation with User Exit 213
BS2000 214
OS/390 or z/OS 215
VM/ESA or z/VM 216
VSE/ESA 217

6. ADADCK : CHECK DATA STORAGE 219

Functional Overview 219
DSCHECK : Check Data Storage 220
Optional Parameters and Subparameters 220
Examples 221
JCL/JCS Requirements and Examples 222
BS2000 222
OS/390 or z/OS 223

VM/ESA or z/VM 225
VSE/ESA 226

7. ADADEF : DEFINE A DATABASE 227

Functional Overview 227
Database Components 227
Checkpoint File 227
DEFINE : Defining a Database and Checkpoint File 228
Syntax 228
Essential Parameters 229
Optional Parameters 231
Examples 236
MODIFY : Change Encodings 237
Optional Parameters 238
Examples 239
NEWWORK : Defining a Work File 240
Essential Parameter 240
Optional Parameters 241
Example 241
JCL/JCS Requirements and Examples 241
BS2000 242
OS/390 or z/OS 245
VM/ESA or z/VM 247
VSE/ESA 249

8. ADAFRM : FORMAT 251

Functional Overview 251
Statement Restrictions 251
Formatting Operation 251
Formatting Modes 252
Syntax 252
Essential Parameter 254
Optional Parameters 254
Examples 255

JCL/JCS Requirements and Examples 257
BS2000 257
OS/390 or z/OS 259
VM/ESA or z/VM 261
VSE/ESA 262

9. ADAICK : CHECK INDEX AND ADDRESS CONVERTER 265

Functional Overview 265
Summary of Functions 266
ACCHECK : Check Address Converter 267
Essential Parameter 267
Optional Parameters 267
ASSOPRINT : Print/Dump Associator Blocks 268
Essential Parameter 268
Optional Parameter 268
BATCH : Set Printout Width to 132 Characters Per Line 269
Optional Parameter 269
DATAPRINT : Print/Dump Data Storage Blocks 270
Essential Parameter 270
Optional Parameter 270
DSCHECK : Print/Dump Content of Data Storage Record 271
Essential Parameter 271
Optional Parameters 271
DUMP : Suspend Dump Suppression 272
Optional Parameter 272
FCBPRINT : Print/Dump File Control Block 273
Essential Parameter 273
Optional Parameters 273
FDTPRINT : Print/Dump Field Definition Table 274
Essential Parameter 274
Optional Parameters 274
GCBPRINT : Print/Dump General Control Block 275
Optional Parameter 275

ICHECK : Check Index and Address Converter 276
Essential Parameter 276
Optional Parameters 276
INT : Cancel Formatted Printout Suppression 277
Optional Parameter 277
NIPRINT : Print/Dump Normal Index 278
Essential Parameter 278
Optional Parameter 278
NOBATCH : Set Print Width to 80 Characters Per Line 279
Optional Parameter 279
NODUMP : Suppress Dumps 280
Optional Parameter 280
NOINT : Suppress Formatted Printout 281
Optional Parameter 281
PPTPRINT : Print/Dump Parallel Participant Table 282
Optional Parameters 282
Example Output 283
UIPRINT : Print/Dump Upper Index 284
Essential Parameter 284
Optional Parameters 284
Examples 285
JCL/JCS Requirements and Examples 286
Collation with User Exit 286
BS2000 286
OS/390 or z/OS 288
VM/ESA or z/VM 289
VSE/ESA 290

10. ADAINV : INVERT 291

Functional Overview 291
COUPLE : Define a File-Coupling Descriptor 292
Essential Parameters 292
Optional Parameters 293
Example 294
Temporary Space for File Coupling 295

Associator Coupling Lists 296
Space for Coupling Lists 297
Space Allocation 298
INVERT : Create a Descriptor 298
Essential Parameters 299
Optional Parameters and Subparameters 299
Space Allocation for the INVERT Function 302
Examples 302
JCL/JCS Requirements and Examples 303
Collation with User Exit 303
BS2000 304
OS/390 or z/OS 307
VM/ESA or z/VM 309
VSE/ESA 311

11. ADALOD : LOADER 313

Functional Overview 313
LOAD : Load a File 314
Essential Parameters 315
Optional Parameters and Subparameters 318
Examples 332
LOAD Data and Space Requirements 334
Loading Expanded Files 337
Loading Multiclient Files 340
UPDATE : Add/Delete Records 341
Essential Parameters 342
Optional Parameters and Subparameters 343
Examples 348
Formats for Specifying ISNs 349
UPDATE Data and Space Requirements 351
Mass Updates of Expanded Files 353
Loader Storage Requirements and Use 354
Static Storage 354
Dynamic Storage 354
Temp Dataset Space Usage 355
Sequential Temp Dataset 355

ADALOD Space/Statistics Report 356
JCL/JCS Requirements and Examples 358
Collation with User Exit 358
BS2000 359
OS/390 or z/OS 363
VM/ESA or z/VM 366
VSE/ESA 369

12. ADAMER : ADAM ESTIMATION 373

Functional Overview 373
Estimate ADAM Access Requirements 374
Essential Parameters 374
Optional Parameters 375
Examples 377
ADAMER Output Report Description 378
JCL/JCS Requirements and Examples 379
BS2000 379
OS/390 or z/OS 380
VM/ESA or z/VM 381
VSE/ESA 382

APPENDIX A : ADABAS SEQUENTIAL FILES 385

Sequential File Table 385
Operating System Dependencies 389
BS2000 Systems 389
OS/390 or MVS/ESA Systems 394
VM/ESA Systems 395
VSE/ESA Systems 396

APPENDIX B :
PROCEDURES FOR VSE/ESA EXAMPLES 401

APPENDIX C : SUPPLIED UES ENCODINGS 403

Interoperable Encodings 404
Single-Byte Character Sets (Latin–1) 404
Single-Byte Character Sets (Non-Latin–1) 405
Double- and Multiple-Byte Character Sets 406
Coexistent Encodings 407
Single-Byte Character Sets 407
Double- and Multiple-Byte Character Sets 408

INDEX 411

GENERAL INFORMATION

Note:
Dataset names starting with DD are referred to in Adabas manuals with a slash separating the DD from the remainder of the dataset name to accommodate VSE/ESA dataset names that do not contain the DD prefix. The slash is not part of the dataset name.

About This Manual

The *Adabas Utilities Manual* comprises two volumes. The utilities are ordered alphabetically by utility name over both volumes: volume 1 contains chapters for the utilities ADAACK through ADAMER; volume 2 contains chapters for the utilities ADAORD through ADAZAP. The chapters are also numbered consecutively over both volumes.

This chapter and appendixes A, B, and C are included in both volumes for your convenience.

Each Adabas utility is described in a separate chapter. For a single-function utility, the chapter begins with a syntax diagram showing the utility statement and all possible parameters. Chapters for utilities with multiple functions begin with a brief overview of the functions, followed by the individual function syntax diagrams and descriptions.

Each function description contains

- syntax diagram with all parameters;
- individual parameter descriptions describing coding rules, restrictions, and defaults; and
- utility function examples.

Following the function descriptions are job control examples for the BS2000, z/OS and OS/390, z/VM and VM/ESA, and VSE/ESA operating systems.

Utility Control Statements

Control Statement Syntax

Utility control statements have the following format:

```
utility  function  parameter-list
```

—where

utility	is the name of the utility to be executed. Examples of utility names include ADAORD, ADADBS, ADAINV.
function	is the name of the specific utility operation to be executed. For example: ADAORD REORDATA ADADBS ADD ADAINV COUPLE Most single-function utilities (ADASEL, ADAULD, etc.) whose function is implicit have either no function value or an optional one.
parameter-list	is a list of parameters following the function. Parameters in the list are almost always keywords with the format "PARAMETER=value ". A parameter may have one or more operands, and keyword parameters may be specified in any order. Most parameters require that you select or otherwise specify an operand value. Some operands are positional (value1,value2,...,valuex), meaning that the values must be in a certain order as described in the text. All parameters must be separated by commas.

Parameter List

In the statement syntax descriptions in this manual, parameters are listed vertically; that is, "stacked". Each list shows all possible parameters, from which one or more can (or must) be specified. Although parameters in the list must be separated by commas, these commas are omitted in the syntax statements when the parameters are stacked.

Required and Optional Parameters

Parameters can be required or optional. Optional parameters (or parameter groups) are shown in square brackets [ ]. For example, the ADADBS CHANGE function has the following parameters:

```
ADADBS CHANGE  FILE=file-number
               FIELD='field-name'
               LENGTH=new-length
               [PASSWORD='password']
               [TEST]
```

The ADADBS CHANGE operation changes the standard length of an Adabas field. The FILE that contains the field to be changed must be specified, as well as the FIELD and the new standard LENGTH for the field. The PASSWORD parameter must be specified only if the specified file is password-protected. You can choose to specify the TEST parameter or not; there are no circumstances where it is required.

Some optional parameters have default values that are used if the parameter is not specified. Default values are indicated by underlined values (see the section **Default Parameter Values** on page 9).

Subparameters

Indented parameter lists are **subparameters**; they cannot be specified unless the parameter above the list is also specified. The following example shows the syntax for subparameters of the FILE parameter:

```
utility function   [FILE=file-number]
                       DATAPFAC=padding-factor
                       DSRABN=startng-rabn
                       [DSSIZE=size]
                       [NU]
```

FILE is an optional parameter. If FILE= is actually specified, then DATAPFAC and DSRABN must also be specified. Note that any subparameter that is not enclosed in square brackets is required, and listed first. The remaining parameters indented under FILE and enclosed in square brackets can optionally be specified when the FILE parameter is specified.

Mutually Exclusive Parameters

Parameters that cannot be specified with each other are called **mutually exclusive** parameters, and are shown in curly brackets, otherwise known as braces { }.

When parameters are stacked and enclosed in braces, one parameter must be specified, but only one can be specified. For example, in the following parameter list:

ADADBS INCREASE { **ASSOSIZE=**size | **DATASIZE=**size }
[**TEST**]

—either the parameter ASSOSIZE or DATASIZE must be specified when executing ADADBS INCREASE; but not both.

Optionally, the parameter TEST may be specified with either ASSOSIZE or DATASIZE.

The following example shows just one of the parameter options of ADAREP REPORT:

ADAREP [**REPORT**] [{ **LIMCOUNT** | **NOCOUNT** }]

In this case, the braces { } within the square brackets [ ] indicate that, if you choose to specify the parameter option at all, either LIMCOUNT or NOCOUNT may be specified.

ADAREP reads the address converter to determine the value for RECORDS LOADED for a file. For very large records, this can result in a large amount of I/O activity.

- If LIMCOUNT is specified, ADAREP checks the value for TOPISN for the file. If TOPISN is greater than 1000, “NOT COUNTED” appears under RECORDS LOADED.
- If NOCOUNT is specified, no value appears under RECORDS LOADED for any file.
- If neither LIMCOUNT nor NOCOUNT is specified, ADAREP compiles the exact value for RECORDS LOADED for each file.

Repeating Parameters and Values

Some parameters can be repeated, and some parameters can specify multiple values. An ellipsis (...) following the parameter or value indicates that it may be repeated. A comma-ellipsis (,...) following the parameter or value indicates that it may be repeated and that repetitions must be separated by a comma.

For longer parameters, parameter combinations, or other lengthy values that can be repeated, the repeatable portion of the syntax is enclosed in braces, followed immediately by a separating comma and ellipsis.

In the following example, FILECRIT and FILEVAL use the '{ },...' notation to indicate that multiple complex criteria and value entries may be specified, respectively.

```
ADALOD LOAD     FILE=file-number [ , file-type ]
                DSSIZE=size
                MAXISN=maximum-number-of-records
                SORTSIZE=size
                TEMPSIZE=size
                FILECRIT={ '{file-selection-criteria}, ...', FILEVAL='value, ... ' |
                                'ISN', MINISN=lowest-allocated-isn  }
```

Some utilities allow a function or parameter itself to be specified multiple times, usually each on separate function statements. For instance, ADACMP field definition statements (FNDEF) might be specified as follows:

```
ADACMP COMPRESS NUMREC=1000
ADACMP FNDEF='01,AA,8,B,DE'
ADACMP FNDEF='01,BA,6,A,NU'
ADACMP FNDEF='01,BB,8,P,NU'
ADACMP FNDEF='01,AD,1,A,FI'
ADACMP SUBDE='CA=BA(1,3)'
```

Control Statement Rules

The following rules apply for the construction of utility control statements:

1. Each control statement must contain a utility name in positions 1–6.
2. The utility function name follows the utility name, separated by at least one space.
3. Keyword parameter entries and multiple values within keyword entries must be separated by commas.
4. No space is permitted before or after "=".
5. The comma following the last parameter entry of a statement is optional.
6. Control statement processing ends with position 72 or when a space is encountered after the beginning of the parameter list. Entries made in positions 73–80 are not processed.
7. A statement that contains an asterisk "*" in position 1 is read as a comment and is not processed.
8. Control statements are continued by specifying the extra parameters on a new statement following (and separated by at least one space from) the utility name in positions 1–6.

Parameter Values

Variable values actually specified following the equals "=" sign in parameters (represented by italicized labels in the preceding examples and elsewhere in this manual) have the following syntax:

```
parameter = value
parameter = value-list
parameter = value-range
```

—where "value" is as described in the following sections. "Value-list" and "value-range" are variations of "value", and are allowed either in place of or with "value", depending on the individual parameter rules as described in the text.

value

"Value" may consist of a number or a string of alphanumeric or hexadecimal characters. In some optional keyword parameters, a default value is assumed if the parameter is not specified; see the section **Default Parameter Values** on page 9 for more information.

Alphanumeric Values

Alphanumeric values are specified in one of the following ways:

If the value comprises . . .	Apostrophes around it are . . .
only upper- or lowercase letters, numeric digits and minus (–)	optional
any other characters including an apostrophe itself (which must be entered twice)	required

Numeric Values

Numeric values are specified as follows:

If the value represents . . .	Specify . . .
a number of either blocks or cylinders	the letter B must immediately follow the value if blocks are being specified; otherwise, cylinders are assumed: **SIZE=200B (200 blocks)** **SIZE=200 (200 cylinders)**
an Adabas file	a one- to four-digit number (leading zeros permitted): **FILE=3** **FILE=03** **FILE=162**
a device type	a four-digit number corresponding to the model number of the device type to be used: **DEVICE=3380**
a field name or descriptor	a two-character field name corresponding to the field name or descriptor: **FIELD1=NA**

Hexadecimal values are accepted if this is specified in the parameter description. Hexadecimal values must be within apostrophes following the indicator X:

X'0002DC9F'

value-list

value,... (numeric values)

BITRANGE=2,10,2

or

’value,...’ (alphanumeric values)

UQDE=’AA,AC,AE’

value-range

value – value,...

ISN=600–900,1000–1200

Individual values within a value list or value range may be positional if they relate to values specified on corresponding parameters. For example:

ADADBS UNCOUPLE FILES=13,20,PASSWORD=’PW13,PW20’

—instructs the ADADBS UNCOUPLE function to uncouple files 13 and 20, which are password-protected.

The passwords (specified by the PASSWORD parameter) must be in the same order as their corresponding files in the FILES parameter.

If file 13 is **not** password-protected, either the PASSWORD parameter must be specified with a “placeholder” comma as shown below

... **PASSWORD=’,PW20’**

—to position the password “PW20” to the corresponding position of file 20 in the FILES value list, or FILES must specify file 20 first.

Default Parameter Values

In the ADADBS ADD function, for example, the optional ASSODEV parameter can specify the device type where the ADADBS ADD function allocates the new dataset space; if ASSODEV is not specified, a currently effective device type (the “default”) is assumed—in this case, a 3380 disk device. Where appropriate, default values or settings are shown underlined in the syntax statement:

[**FILE=** { file-number | **1** }]

Otherwise, the defaults are explained in the accompanying parameter descriptions. If desired, the default value can be explicitly entered (in this example, FILE=1).

If the default is a value originally established by some other control statement, the parameter name is shown in underlined italics as the default:

[**TTSYN=**{ seconds | ADARUN-tt *}*]

Here, the default is the value already set by the TT parameter in the Adabas runtime control (ADARUN) statement, which is in effect if TTSYN is not specified. The value “tt” should not actually be entered.

ADAACK : CHECK ADDRESS CONVERTER

Functional Overview

ADAACK checks the address converter for a specified file, a range of files, or all files and/or for a specified ISN range or all ISNs. It is used in conjunction with ADAICK.

ADAACK checks each address converter element to determine whether the Data Storage RABN is within the used portion of the Data Storage extents specified in the file control block (FCB).

ADAACK checks the ISN for each record in each Data Storage block (within the specified ISN range) to ensure that the address converter element for that ISN contains the correct Data Storage RABN. This is done in the following way:

1. Main memory is allocated for the specified range of ISNs (number of ISNs, times 4). If no range is specified, the entire range (MINISN through TOPISN) is checked.

   The address converter is read from the database into this area in memory.

2. Each used Data Storage block (according to the Data Storage extents in the FCB) is read and checked against the address converter in memory. Each ISN in the address converter must have exactly one associated Data Storage record.

3. The address converter in memory is checked for ISNs that did not occur in Data Storage.

For large files, ADAACK may run for a long time. ADAACK prints a message line after every 20 Data Storage blocks processed.

Run time is not affected by the ISN range, since all used Data Storage blocks are read.

Notes:

1. *ADAACK does not require the Adabas nucleus to be active.*
2. *A pending autorestart condition is ignored.*
3. *ADAACK does not synchronize with the nucleus in case of parallel updating.*
4. *This utility should be used only for diagnostic purposes.*

ADAACK returns a condition code 8 if any errors occur.

ACCHECK : Check Address Converter Against Data Storage

ADAACK ACCHECK [**FILE=**{file | file – file | all-files}]
[**ISN=**{isn – isn | all-isns}]
[**NOUSERABEND**]

Optional Parameters

FILE : Files to be Checked

The file, single range of files, or all files to be checked. By default, all files in the database are checked.

ISN : ISN Range to be Checked

A range of ISNs or all ISNs to be checked. By default, the entire range MINISN through TOPISN is checked.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

Examples

Example 1:

ADAACK ACCHECK

Check all files in the database.

Example 2:

ADAACK ACCHECK FILE=12, ISN=1–8000

Check ISNs 1 through 8000 for file 12.

Example 3:

ADAACK ACCHECK FILE=8–10

Check all ISNs in files 8 through 10.

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADAACK with BS2000, z/OS or OS/390, z/VM or VM/ESA, and VSE/ESA systems and shows examples of each of the job streams.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations Manual*
ADAACK parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADAACK messages	SYSLST DDDRUCK		*Messages and Codes*

ADAACK JCL Examples (BS2000)

In SDF Format:

```
/.ADAACK LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A A C K ADDRESS CONVERTER CHECK
/REMARK *
/REMARK *
/ASS-SYSLST L.ACK.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAACK,DB=yyyyy,IDTNAME=ADABAS5B
ADAACK ACCHECK FILE=ffff
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAACK LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A A C K ADDRESS CONVERTER CHECK
/REMARK *
/REMARK *
/SYSFILE SYSLST=L.ACK.DATA
/FILE ADAvrs.MOD ,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADAACK,DB=yyyyy,IDTNAME=ADABAS5B
ADAACK ACCHECK FILE=ffff
/LOGOFF NOSPOOL
```

z/OS or OS/390

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADAACK messages	DDDRUCK	printer	*Messages and Codes*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADAACK parameters	DDKARTE	reader	

ADAACK JCL Example (z/OS or OS/390)

```
//ADAACK    JOB
//*
//*   ADAACK:
//*      ADDRESS CONVERTER CHECK
//*
//ACK       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1  <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1  <=== DATA
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAACK,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAACK ACCHECK FILE=ffff
/*
//
```

Refer to ADAACK in the MVSJOBS dataset for this example.

z/VM or VM/ESA

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADAACK messages	DDDRUCK	disk/ terminal/ printer	*Messages and Codes*
ADARUN messages	DDPRINT	disk/ terminal/ printer	*Messages and Codes*
ADARUN parameters	DDCARD	disk/ terminal/ reader	*Operations Manual*
ADAACK parameters	DDKARTE	disk/ terminal/ reader	

ADAACK JCL Example (z/VM or VM/ESA)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPRINT,DSN=ADAACK.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAACK.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNACK.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAACK.CONTROL,MODE=A
ADARUN
```

Contents of RUNACK CONTROL A1:

```
ADARUN PROG=ADAACK,DEVICE=dddd,DB=yyyyy
```

Contents of ADAACK CONTROL A1:

```
ADAACK ACCHECK FILE=ffff
```

VSE/ESA

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
ADAACK messages		printer	SYS009	*Messages and Codes*
ADARUN messages		printer	SYSLST	*Messages and Codes*
ADARUN parameters	 CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAACK parameters		reader	SYSIPT	

***** Any programmer logical unit may be used.

ADAACK JCS Example (VSE/ESA)

See appendix B for descriptions of the VSE/ESA procedures (PROCs).

```
* $$ JOB JNM=ADAACK,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       ADDRESS CONVERTER CHECK
// JOB ADAACK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAACK,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAACK ACCHECK FILE=27
/*
/&
* $$ EOJ
```

Refer to member ADAACK.X for this example.

2

ADACDC : CHANGED-DATA CAPTURE

Functional Overview

The ADACDC utility

- takes as input one or more sequential protection logs; and
- produces as output a delta of all changes made to the database over the period covered by the input protection logs.

"Delta of changes" means that the last change to each ISN in a file that was altered during this period appears on the primary output file.

This output may be used on a regular basis as input for data warehousing population procedures so that the delta of changes to a database is applied to the data warehouse database rather than a copy of the entire database. This affords more frequent and less time consuming updates to the data warehouse, ensuring greater accuracy of the information stored there.

In order to run the ADACDC utility

- an external sorter must be available and installed as the standard sorter in the operating system. See page 27 for more information.
- the external sorter must have access to the database's Associator containing the FDTs of the files for which records are to be processed.

Phases of Operation and Resulting Files

ADACDC processes sequential protection logs in two phases. You can execute phase 1 and phase 2 separately, or both at once (the default):

- If phase 2 is being run separately or both phases are being completed together, the data is decompressed and written to the **primary output file**.
- If only phase 1 is being executed, the data is written to an **extract file**. This extract file may then be processed multiple times by a phase 2 operation to decompress the records and write to primary output files.

The extract file contains data records in compressed format whereas the primary output file contains records in decompressed format. Refer to the chapter **ADACMP (Compress – Decompress)** in this manual for more information about these formats.

The primary output file and the extract file are standard operating-system-dependent files that can handle variable length records.

Phase 1 and the Extract File

During phase 1, updates from the protection logs are analyzed and prefixed with a standard structure called the CDCE. The format of each record on the file is a constant CDCE prefix followed by the compressed record information. These records are passed to an external SORT routine to establish the most recent update for each ISN on a file. Only the last change for a given file and ISN combination is written.

The extract file created when phase 1 is run separately makes it possible to process the PLOG data once and then optionally produce multiple primary output files from it based, for example, on file selection criteria. The option is useful if different file changes are required for different purposes.

When the phase 1 process is being run, the extract file is opened for output. As records are output from the SORT processing, the latest update for each file and ISN combination is written to the extract file if

- the update was performed by an ET user and belongs to a completed transaction; or
- the update was performed by an EXU user and belongs to a completed command; or
- NOET is specified.

All other updates for the file and ISN combination for that period are discarded if there are no controlled utility operations against that file (see the section **Checkpoints Written to the Primary Output File** on page 21).

Note:
It is possible to have duplicate file and ISN combinations on the file if the ADACDC user exit (described later) adds records with file and ISN combination that already exist. A record added or modified by the user exit is so marked in the CDCE structure.

Phase 2 or Both and the Primary Output File

The primary output file is used when both stages of ADACDC are run together, or for phase 2 processing only.

- If both phases are run together, the primary output file is opened and created directly using the output from the SORT processing. In this case, processing occurs as for the extract file for phase 1 processing.

- If only phase 2 is run, the primary output file is created using input from the extract file.

  The format of each record on the file is a constant CDCO prefix followed by the decompressed record information. If for some reason the record cannot be decompressed, a warning message is issued and the compressed record is written to the primary output dataset. A flag in the CDCO structure informs a user program when decompression for the record has failed.

  Note:
  It is possible to have duplicate file and ISN combinations on the file if the ADACDC user exit (described later) adds records with file and ISN combinations that already exist. A record added or modified by the user exit is so marked in the CDCO structure.

Checkpoints Written to the Primary Output File

The primary objective of the ADACDC utility is to provide an output dataset containing the most recent changes for each ISN in a file that has been modified for the period concerned.

Apart from simple changes to a file, some utility operations executed against a file may fundamentally affect its contents. For example, if the file is deleted, simply providing the last updates for ISNs in the file does not accurately reflect the state of the file since all ISNs have been deleted.

For this reason, the following checkpoints are recorded and written to the primary output file as appropriate with the associated indication in the output record:

```
ADASAV RESTORE FILE          File created
ADAORD STORE FILE            File created
ADALOD LOAD FILE             File created
ADALOD UPDATE FILE           File updated
ADADBS DELETE FILE           File deleted
ADADBD REFRESH FILE          File deleted
```

Because these operations can fundamentally impact a file and its appearance, the checkpoint is written to the primary output dataset when it occurs relative to the other updates.

ADACDC retains the last change to all ISNs before each of the above checkpoints. This means that a file and ISN combination could appear multiple times on the primary output file if one or more checkpoints were written to it. This is useful for the many data warehouse packages that may wish to complete their view of a file and maintain a copy of it prior to deletion, re-creation, or mass update.

Primary Input Data

The primary input data comprises sequential protection logs produced either

- by the database directly; or
- by the ADARES PLCOPY of nonsequential protection logs.

ADACDC processes this data to ensure that

- when a new PLOG block is read and the PLOG number is the same, the PLOG block number is 1 greater than the previous PLOG block number.
- when the PLOG number itself changes, the new PLOG number is higher than the previous PLOG number and the new PLOG block number is 1.

Note:
When the PLOG number changes and the difference between the PLOG numbers is greater than 1, a warning message is issued and processing continues as this can legitimately happen if online saves are used.

If any of these checks fail, the utility execution terminates.

Primary Output Data

The primary output data is a sequential file comprising all database records that were added, updated, or deleted during the period covered by the input protection logs.

If a record was changed several times, only its last change appears in the output file. ADACDC employs a SORT process to identify multiple changes to the same record.

Each primary output record comprises a fixed-length record prefix followed by the database record in decompressed form. The decompressed data corresponds in format to the output of the ADACMP DECOMPRESS function (see page 57 for more information).

The primary output record prefix is described by the CDCO DSECT. It has the following structure:

Bytes	Description
0–1	record length (binary)
2–3	set to zeros
4–7	constant ‘CDCO’
8–9	database ID
10–11	file number
12–15	ISN of the updated record
16–19	length of the decompressed data in bytes
20–47	28-byte communication ID of the last user who updated the record
48	change indicator: X‘04’ record added X‘08’ record updated X‘0C record deleted X‘10’ file created X‘14’ file updated X‘18’ file deleted or refreshed
49	flags (independent bit settings): X‘80’ record added by user exit X‘40’ record modified by user exit X‘20’ record still compressed; decompression failed
50	database version indicator
51–67	reserved for future use
68–...	decompressed record data

Transaction File

To maintain input data checking over multiple runs of the utility, ADACDC stores information on the transaction file in a transaction control record containing the last database ID, the PLOG number, and the PLOG block number processed. This information is used to verify the latest input (unless the RESETTXF option is specified – see page 26).

ADACDC actually recognizes two different transaction files: input and output. Both transaction files are standard operating-system-dependent files that can handle variable length records.

Input Transaction File Processing

During the input processing stage, ADACDC processes the input transaction file to the SORT program.

Following the control record on the input transaction file, 0 or more records may be found. These are database updates related to transactions not completed during the last run of the utility. These records are processed again as part of the input as their transactions will normally have been completed in the next sequential protection logs provided to the utility. This is the reason the sequence of protection logs is so important: updates may remain outstanding forever if the correct sequence is not maintained.

The transaction file also records whether the NOET option was specified during the last phase 1 run of the utility. When ADACDC detects that this option has changed from one utility execution to the next, it uses the information from the control record on the input transaction file; however, all transactional information in the other records is ignored. This is due to the fact that changing this option may cause inconsistent data to be written to the primary output file or extract file, as appropriate. ADACDC issues a warning when this happens.

Output Transaction File Processing

Once output processing from the SORT program starts, the input transaction file is closed and the output transaction file is opened. The control record is written to the output transaction file followed by any updates that relate to incomplete transactions or, in the case where the NOET option is in effect or an EXU user is in control, to incomplete commands. The output transaction file is closed once processing is complete.

Using a Single Transaction File

It is possible to use the same file as both the input and output transaction file; however, if the utility fails while writing to the output transaction file (that is, at any time during the output processing of the SORT utility), the input transaction file will no longer exist and therefore, rerunning the utility will yield a different result.

For this reason, the transaction file must be backed up prior to the utility run so that it can be restored in the event of a failure.

Alternatively, you could use a facility on your operating system (if available) that produces a new version of a file whenever a program updates the file.

Running the Utility

ADACDC [**FILE=** filelist]
[**NOET**]
[**PHASE=1|2|BOTH**]
[**RESETTXF**]

The first time you run the ADACDC utility, use the following syntax and either not specify or dummy the input transaction file (CDCTXI) to create a valid transaction file for input to subsequent runs:

ADACDC RESETTXF,PHASE=BOTH

The RESETTXF option ignores the absent or dummied input transaction file, reads the primary input data, and produces primary output using the input data.

After the input transaction file has been created during the first run, only the utility name ADACDC is required to run this utility; the PHASE parameter defaults to BOTH. Parameter options are explained in the following sections.

Optional Parameters

FILE : Files Processed

Use the FILE parameter to limit the file(s) processed by the utility and written to the output file:

- For phase 1 operation, only records relating to the files specified are written to the extract file.
- For phase 2 and BOTH operations, only records relating to the files specified are written to the primary output file.

Note:
Clearly, files required for phase 2 processing must have been specified on the previous phase 1 operation that created the input extract file.

When this parameter is not specified, all files are processed by the utility.

NOET : Bypass ET Processing

ADACDC normally accepts for processing only those records that are part of completed transactions or, in the case of EXU users, records that are part of completed commands.

Use the NOET option to bypass this transaction processing when PHASE=1 or PHASE=BOTH. NOET has no effect when PHASE=2 because the input is the extract file from phase 1 which has already processed the protection log (PLOG) input with or without the NOET option.

When NOET is specified, any update made to the database is processed and written to the extract file (PHASE=1) or primary output file (PHASE=BOTH) as soon as it is encountered on the PLOG.

Warning:
Specifying this option may result in updates recorded on the primary output file that are related to transactions that were not complete at the end of the input PLOG.

PHASE : Execution Phase

The PHASE parameter determines the input the utility requires and the output it produces:

- PHASE=1 reads the sequential PLOG input and produces an interim extract file for later processing by a phase 2 step.
- PHASE=2 reads an extract file produced by a previously executed phase 1 step and produces a primary output file containing the delta of changes made to the file.
- PHASE=BOTH (the default) reads the sequential PLOG input and produces the primary output file containing the delta of changes directly without reading or writing an extract file.

Refer to the section **Phases of Operation and Resulting Files** on page 19 for more information.

RESETTXF : Reset Input Transaction File

ADACDC checks the primary input data to the utility to ensure that the PLOGs are read in sequence, by PLOG block and PLOG number. If these checks fail, the utility execution terminates.

To maintain the checks over multiple runs of the utility, ADACDC maintains input and output transaction files. These files also track record updates related to incomplete transactions or, in the case of EXU users, incomplete commands from one utility execution to the next. Normally, such incomplete transactions or commands are completed in the next sequential PLOGs provided to the utility.

However, if the need arises to process PLOGs out of sequence and the information in the transaction file can be safely removed, the RESETTXF option can be used to reset the transaction file so that the checks are bypassed and all outstanding transaction or command data is ignored for a given run. ADACDC ignores information on the input transaction file and writes the output transaction file at end of job.

Warning:
If the sequence of PLOGs is interrupted, record updates related to incomplete transactions recorded in the transaction file may remain outstanding indefinitely.

Operating System Considerations

For its sort requirements, the ADACDC utility uses a standard sort function installed in the operating system. The following additional considerations should be taken into account for each operating system.

z/OS or OS/390

No additional job steps are required by ADACDC when the sort function is invoked. However, depending on the amount of data to be sorted, the ADACDC job step may require additional sort-related DD statements for work files or for other sort-specific facilities. Refer to the sort documentation for more details.

Note:
A sort package generally supplies summary information when a SYSOUT DD statement is specified.

When ADACDC invokes sort, it expects by default to transfer control to a load module named ‘SORT’. If the sort module has a different name, you must reassemble and link the Adabas options module ADAOPD, specifying the name of the external sort program as follows:

1. Modify the OPDOS member, specifying the name of the sort program in parameter SORTPGM=.
2. Modify and run member ASMLOPD to assemble and link the module ADAOPD.

VSE/ESA

Whenever an external sort may be called, an ADACDC utility job must reserve space in the partition area. The EXEC statement must therefore specify the SIZE parameter as either

```
// EXEC ADARUN,SIZE=(ADARUN,128K)
```

or

```
// EXEC ADARUN,SIZE=(AUTO,128K)
```

No additional job steps are required by ADACDC when the sort function is invoked. However, depending on the amount of data to be sorted, the ADACDC job step may require additional sort-related DLBL statements for work files or for other sort-specific facilities. Refer to the sort documentation for more details.

When ADACDC invokes sort, it expects by default to transfer control to a load module named 'SORT'. If the sort module has a different name, the Adabas options module ADAOPD must first be reassembled and relinked with the correct name of the sort module in parameter SORTPGM. See the section **Modify, Assemble, and Link the Adabas Options Table** in the chapter **VSE/ESA Systems Installation** of the *Adabas Installation Manual for VSE/ESA*.

BS2000

The Siemens external sort may be called for large sort operations. The following job cards are required.

```
/SET-FILE-LINK BLSLIBnn,$.SORTLIB
/SET-FILE-LINK SORTWK1,#SORTWK,BUF-LEN=STD(2),OPEN-MODE=OUTIN
/CREATE-FILE #SORTWK,PUB(SPACE=(&PRIM,&SEC))
/START-PROGRAM .....,RUN-MODE=ADVANCED,ALT-LIBRARY=YES
```

—where

nn	is a value between 00 and 99
#SORTWK	was created with the BS2000 command
&PRIM	is the number of primary PAM pages to allocate
&SEC	is the number of secondary PAM pages to allocate

Note:
The size of the SORTWK1 file depends on the amount of data to be sorted.

The ADACDC User Exit

ADACDC calls a user exit at various points in its processing, providing you with the opportunity to intercede in that processing.

Installing the Exit

To install the user exit

1. Compile the user exit you wish ADACDC to use as module name ADACDCUX.
2. Make the module available to the ADACDC utility.

A sample user exit called ADACDCUX is provided on the source dataset. The only function of the sample is to show you how to add, delete, and update records using the user exit interface.

User Exit Interface

The user exit is called with the following registers set:

R1 –> user parameter list

R13 –> standard 72-byte register save area

R14 –> return address

R15 –> entry point

The user parameter list contains two pointers:

- the first to the ADACDC user exit parameter list mapped by the CDCU DSECT; and
- the second to the record area for the user exit where the record being processed is passed as appropriate.

The action to be performed is indicated in the CDCUFUNC field whereas the action the user exit directs ADACDC to take on return is indicated using the CDCURESP field.

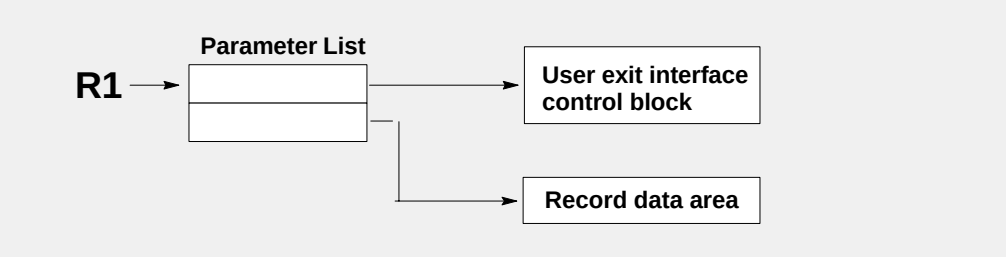


Figure 2-1: ADACDC User Exit

The structure of the ADACDC user exit interface control block (CDCU DSECT) is as follows:

Bytes	Description
0–3	constant ‘CDCU’
4–7	available for use by user exit
8–11	length of record in second parameter
12	function identifier:
	X‘00’ initialization
	X‘04’ before pass to SORT input
	X‘08’ before write to extract file
	X‘0C’ before write to primary output file
	X‘10’ termination
13	response code from user exit:
	X‘00’ normal processing
	X‘04’ ignore this record
	X‘08’ record has been updated
	X‘0C’ insert new record
14–31	reserved for future use

User Exit Calls

The following subsections describe the calls made to the user exit and their purpose.

Initialization Call (CDCUFUNC=CDCUINIT)

During initialization, ADACDC calls the user exit so that it can set up any areas it requires for future processing. The CDCUUSER field is provided in the CDCU for anchoring a user control block, if appropriate.

The record area pointer points to data that has no relevance for this call.

Termination Call (CDCUFUNC=CDCUTERM)

During termination, ADACDC calls the user exit so that it can close any open files or clean up any areas still outstanding after ADACDC execution. For example, if an anchor pointer was set in CDCUUSER, this area could be freed and the CDCUUSER field set to nulls.

The record area pointer points to data that has no relevance for this call.

SORT Input Call (CDCUFUNC=CDCUINPT)

ADACDC calls the user exit before a record is passed to the SORT routine as input.

The record area pointer points to the compressed data record to be returned prefixed by the CDCE control block.

The exit may elect to

- continue processing normally;
- request that the record be ignored;
- update the current record; or
- add a record, in which case the record pointed to by the record area pointer is passed to the SORT routine. Thereafter, each time the exit is called, the same record is presented again until
  - normal processing continues; or
  - the record is ignored or updated, at which time processing continues with the next input record.

Extract Output Call (CDCUFUNC=CDCUWRTE)

ADACDC calls the user exit before a record is written to the extract file during phase 1 processing. This exit point is **only** called during phase 1 processing and has no relevance in other cases.

The record area pointer points to compressed the data record to be written prefixed by the CDCE control block.

The exit may elect to

- continue processing normally;
- request that the record be ignored;
- update the current record; or
- add a record, in which case the record pointed to by the record area pointer on return is written to the extract file. Thereafter, each time the exit is called, the same record is presented again until
  - normal processing continues; or
  - the record is ignored or updated, at which time processing continues with the next record to be written to the extract file.

Primary Output Call (CDCUFUNC=CDCUWRTO)

ADACDC calls the user exit before a record is written to the primary output file. This exit point is **not** called during phase 1 processing and has no relevance in this case.

The record area pointer points to the decompressed data record to be written prefixed by the CDCO control block.

The exit may elect to

- continue processing normally;
- request that the record be ignored;
- update the current record; or
- add a record, in which case the record pointed to by the record area pointer on return is written to the primary output file. Thereafter, each time the exit is called, the same record is presented again until
  - normal processing continues; or
  - the record is ignored or updated, at which time processing continues with the next record to be written to the output file.

Updating or Adding Records

Consider the following points when updating or adding records from the exit:

- The CDCELEN/CDCERECL fields must correctly reflect the length of data following the CDCEDATA field for the input and write extract exit points.
- The CDCOLEN/CDCORECL fields must correctly reflect the length of data following the CDCODATA field for the input and write extract exit points.
- For the input exit point, records added should have a unique ISN if no ISN update is to be replaced.
- For the input exit point where an ISN is to be replaced, the last occurrence of the ISN should be updated or the replacement record for the ISN should be added after all occurrences for the ISN have been seen.
- When adding records at the extract or primary output exit points, be aware that if file and ISN combinations are duplicated, the duplicated information is written to the primary output file which may confuse processing routines for that file.

Examples

ADACDC RESETTXF,PHASE=BOTH

Ignoring any information on the input transaction file, reads the primary input data and produces primary output using the input data.

Use this syntax and either remove or dummy the input transaction file (CDCTXI) the first time you run the utility to create a valid transaction file for input to subsequent runs.

ADACDC PHASE=1
ADACDC FILE=20
ADACDC FILE=40–50

Reads the primary input data and processes data only for files 20 and 40 to 50 inclusive. The latest updates for each ISN on those files are written to the extract file.

ADACDC PHASE=2
ADACDC FILE=44–47

Reads a previously created extract file and writes all records for files 44, 45, 46, and 47 to the primary output file.

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADACDC with BS2000, z/OS or OS/390, z/VM or VM/ESA, and VSE/ESA and shows examples of each of the job streams.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	required to read the GCB and FDT entries
Protection log	DDSIIN/ DDSIINnn	tape/disk	sequential log (not required when PHASE=2)
Extract file	CDCEXT	tape/disk	output or input extract file (not required when PHASE=BOTH)
Input transaction file	CDCTXI	tape/disk	not required when RESETTXF is specified or when PHASE=2
Output transaction file	CDCTXO	tape/disk	not required when PHASE=2
Primary output file	CDCOUT	tape/disk	not required when PHASE=1
ADARUN parameters	SYSDTA/ DDCARD	disk/terminal/ reader	*Operations Manual*
ADACDC parameters	SYSDTA/ DDKARTE	disk/terminal/ reader	*Utilities Manual*
ADARUN messages	DDPRINT	disk/terminal/ printer	*Messages and Codes*
ADACDC messages	DDDRUCK	disk/terminal/ printer	*Messages and Codes*

ADACDC JCL Example (BS2000)

```
     /.ADACDC LOGON
     /REMA  ADACDC : CAPTURE DELTA CHANGES
     /REMA
     /ASS-SYSOUT EXAMPLE.ADACDC.SYSOUT
     /MODIFY-TEST-OPTION DUMP=YES
     /DELETE-FILE EXAMPLE.OUTPUT.TRANS.FILE
     /SET-JOB-STEP
     /DELETE-FILE EXAMPLE.OUTPUT.PRIMARY.FILE
     /SET-JOB-STEP
     /CREATE-FILE EXAMPLE.OUTPUT.TRANS.FILE,PUB(SPACE=(48,48))
     /CREATE-FILE EXAMPLE.OUTPUT.PRIMARY.FILE,PUB(SPACE=(48,48))
     /ASS-SYSDTA *SYSCMD
     /SET-FILE-LINK  DDASSOR1,EXAMPLE.DByyyyy.ASSOR1
     /SET-FILE-LINK  DDSIIN,EXAMPLE.DByyyyy.PLOG000
     /SET-FILE-LINK  DDSIIN01,EXAMPLE.DByyyyy.PLOG001
     /SET-FILE-LINK  DDSIIN02,EXAMPLE.DByyyyy.PLOG002
     /SET-FILE-LINK  DDSIIN03,EXAMPLE.DByyyyy.PLOG003
     /SET-FILE-LINK  CDCTXI,EXAMPLE.INPUT.TRANS.FILE
     /SET-FILE-LINK  CDCTXO,EXAMPLE.OUTPUT.TRANS.FILE
     /SET-FILE-LINK  CDCOUT,EXAMPLE.OUTPUT.PRIMARY.FILE
     /SET-FILE-LINK  DDDRUCK,EXAMPLE.ADACDC.DRUCK
     /SET-FILE-LINK  DDPRINT,EXAMPLE.ADACDC.PRINT
     /SET-FILE-LINK  DDLIB,ADABAS.Vvrs.MOD
     /START-PROGRAM *M(ADABAS.Vvrs.MOD,ADARUN)
     ADARUN
PROG=ADACDC,MODE=MULTI,IDTNAME=xxxxxxxx,DEVICE=dddd,DBID=yyyyy
     ADACDC
PHASE=BOTH,FILE=1,10,20-30
     /LOGOFF SYS-OUTPUT=DEL
NOSPOOL
```

z/OS or OS/390

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	required to read the GCB and FDT entries
Protection log	DDSIIN	tape / disk	sequential log (not required when PHASE=2)
Input transaction file	CDCTXI	tape / disk	not required when RESETTXF is specified or when PHASE=2
Output transaction file	CDCTXO	tape / disk	not required when PHASE=2
Extract file	CDCEXT	tape / disk	output or input extract file (not required when PHASE=BOTH)
Primary output file	CDCOUT	tape / disk	not required when PHASE=1
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADACDC parameters	DDKARTE	reader	*Utilities Manual*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADACDC messages	DDDRUCK	printer	*Messages and Codes*

ADACDC JCL Example (z/OS or OS/390)

Refer to ADACDC in the MVSJOBS dataset for this example.

```
//ADACDC    JOB
//*
//*    ADACDC : CAPTURE DELTA CHANGES
//*
//CDC      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDSIIN    DD   DSN=EXAMPLE.DByyyyy.PLOG(-3),DISP=SHR  <== PLOG TAPE
//          DD   DSN=EXAMPLE.DByyyyy.PLOG(-2),DISP=SHR  <== PLOG TAPE
//          DD   DSN=EXAMPLE.DByyyyy.PLOG(-1),DISP=SHR  <== PLOG TAPE
//          DD   DSN=EXAMPLE.DByyyyy.PLOG(0),DISP=SHR  <== PLOG TAPE
//CDCTXI    DD   DSN=EXAMPLE.input.trans.file,DISP=SHR
//CDCTXO    DD   DSN=EXAMPLE.output.trans.file,DISP=OLD
//CDCOUT    DD   DSN=EXAMPLE.output.primary.file,DISP=OLD
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADACDC,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADACDC PHASE=BOTH,FILE=1,10,20-30
/*
//
```

z/VM or VM/ESA

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	required to read the GCB and FDT entries
Protection log (PLOG)	DDSIIN	disk / tape	sequential log (not required when PHASE=2)
Input transaction file	CDCTXI	disk / tape	not required when RESETTXF is specified or when PHASE=2
Output transaction file	CDCTXO	disk / tape	not required when PHASE=2
Extract file	CDCEXT	disk / tape	output or input extract file (not required when PHASE=BOTH)
Primary output file	CDCOUT	disk / tape	not required when PHASE=1
ADARUN parameters	DDCARD	disk/terminal/ reader	*Operations Manual*
ADACDC parameters	DDKARTE	reader	*Utilities Manual*
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADACDC messages	DDDRUCK	printer	*Messages and Codes*

ADACDC JCL Example (z/VM or VM/ESA)

```
/*                                                    */
/*       ADACDC JCL Example (VM/ESA)                  */
/*                                                    */
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
/*                                                    */
DATADEF DDSIIN,DSN=ADACDC.PLOG,MODE=A
DATADEF CDCTXI,DSN=ADACDC.INNPUT,MODE=A
DATADEF CDCTXO,DSN=ADACDC.OUTPUT,MODE=A
DATADEF CDCOUT,DSN=ADACDC.PRIMARY,MODE=A
/*                                                    */
DATADEF DDPRINT,DSN=ADACDC,DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADACDC.DDDRUCK,MODE=A
/*                                                    */
DATADEF DDCARD,DSN=RUNCDC.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADACDC.CONTROL,MODE=A

EXECOS ADARUN
RCODE = RC
EXIT RCODE
```

Contents of RUNCDC CONTROL A1:

```
ADARUN PROG=ADACDC,DEVICE=dddd,DB=yyyyy
```

Contents of ADACDC CONTROL A1:

```
ADACDC PHASE=BOTH,FILE=1,10,20-30
```

VSE/ESA

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	required to read the GCB and FDT entries
Protection log	SIIN	tape disk	SYS010 *	sequential log (not required when PHASE=2)
Input transaction	CDCTXI	tape disk	SYS015 *	not required when RESETTXF is specified or when PHASE=2
Output transaction	CDCTXO	tape disk	SYS016 *	not required when PHASE=2
Extract	CDCEXT	tape disk	SYS017 *	output or input extract file (not required when PHASE=BOTH)
Primary output	CDCOUT	tape disk	SYS018 *	not required when PHASE=1
ADARUN parameters	– CARD CARD	reader tape disk	SYSRDR SYS000 *	*Operations Manual*
ADACDC parameters	–	reader	SYSIPT	*Utilities Manual*
ADARUN messages	–	printer	SYSLST	*Messages and Codes*
ADACDC messages	–	printer	SYS009	*Messages and Codes*

***** Any programmer logical unit may be used.

ADACDC JCS Example (VSE/ESA)

See appendix B for descriptions of the VSE/ESA procedures (PROCs).

Refer to member ADACDC.X for this example.

```
* $$ JOB JNM=ADACDC,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADACDC
*       CAPTURE DELTA CHANGES
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL SIIN,'EXAMPLE.DByyy.PLOG'
// MTC REW,SYS010
// DLBL CDCTXI,'EXAMPLE.INPUT.TRANS.FILE',,SD
// EXTENT SYS015
// ASSGN SYS015,DISK,VOL=DISK01,SHR
// DLBL CDCTXO,'EXAMPLE.OUTPUT.TRANS.FILE',,SD
// EXTENT SYS016,,,,sssss,nnnnn
// ASSGN SYS016,DISK,VOL=DISK02
// DLBL CDCOUT,'EXAMPLE.OUTPUT.TRANS.FILE',,SD
// EXTENT SYS018,,,,sssss,nnnnn
// ASSGN SYS018,DISK,VOL=DISK04
// EXEC ADARUN,SIZE=ADARUN
ADARUN DBID=yyyyy,DEVICE=dddd,PROG=ADACDC,SVC=xxx,MODE=MULTI
/*
ADACDC PHASE=BOTH,FILE=1,10,20-30
/*
/&
* $$ EOJ
```

3

ADACMP : COMPRESS–DECOMPRESS

Functional Overview

Overview of the COMPRESS Function

The COMPRESS function edits and compresses data records that are to be loaded into the database:

input data → ADACMP COMPRESS → ADALOD LOAD

Input can be data records from

- a physical sequential dataset (fixed- or variable-length records) supplied by the user; or
- an existing Adabas file (that is, from ADACMP DECOMPRESS or ADAULD UNLOAD).

The logical structure and characteristics of the input data are described with **field definition statements**:

- The FNDEF statement is used to define a field (or group of fields).
- The SUBFN and SUPFN statements are used to define a subfield and a superfield, respectively.
- The COLDE, HYPDE, PHONDE, SUBDE, and SUPDE statements are used to define a collation descirptor, hyperdescriptor, phonetic descriptor, subdescriptor and superdescriptor, respectively.

The field definitions provided are used to create the Adabas field definition table (FDT) for the file. It is also possible to use an existing Adabas FDT instead of providing field definitions (see the FDT parameter description on page 61).

If the fields in the input record are to be processed in an order that is different from their position in the input record, and/or if one or more fields are to be skipped, the FORMAT parameter may be used to indicate the order and location of the input fields.

The ADACMP COMPRESS function processes the input data as follows:

- Checks numeric data for validity.
- Removes trailing blanks from alphanumeric fields.
- Removes leading zeros from numeric fields.
- Packs numeric unpacked fields.

Fields defined with the fixed (FI) option are not compressed.

A user exit can be used to further edit the input data.

The output of the ADACMP COMPRESS function that is used as input to the ADALOD utility includes the FDT, compressed records, and on the utility report, the Data Storage space requirement (for the ADALOC DSSIZE parameter) and the temp and sort dataset size estimates (TEMPSIZE and SORTSIZE).

The ADACMP COMPRESS function report also indicates

- the number of records processed;
- the number of records rejected; and
- the compression rate percentage.

A dataset containing rejected records is also produced.

Overview of the DECOMPRESS Function

The DECOMPRESS function decompresses individual files:

input data → ADACMP DECOMPRESS → decompressed records

Input data can be data records from existing Adabas files

- unloaded using the ADAULD (file unload) utility; or
- directly (without separate file unloading).

The INFILE parameter of ADACMP DECOMPRESS is used for Adabas files that are directly decompressed. As part of the decompression process, the target file is unloaded without FDT information, which can save time when decompressing larger files.

The output of the ADACMP DECOMPRESS function includes ISNs if the ISN parameter is specified. The DECOMPRESS output may be used as input to a non-Adabas program or as input to the COMPRESS function, once any desired changes to the data structure or field definitions for the file are completed.

Input Data Requirements

Input Data Structure

ADACMP input data must be in a sequential dataset/file. Indexed sequential and VSAM input cannot be used.

The records may be fixed, variable, or of undefined length. The maximum input record length permitted depends on the operating system. The maximum compressed record length is restricted by the Data Storage block size in use and the maximum compressed record length set for the file (see the MAXRECL parameter, ADALOD utility). The input records can be in either blocked or unblocked format.

The fields in each record must be structured according to the field definition statements provided (or the specified FDT if an existing Adabas FDT is being used). If a user exit routine is used, the structure following user exit processing must agree with the field definitions. Any trailing information contained in an input record for which a corresponding field definition statement is not present is ignored and is not included in the ADACMP output.

Datasets that contain no records are also permitted.

The input dataset can be omitted if the parameter NUMREC=0 is supplied.

Multiple-Value Field Count

The number of values in each record's multiple-value field must either be specified in the field definition statement, or the value count must precede the values in each input record. When specified in the field definition statement, the minimum multiple-value count is 1, and the maximum is 191. When the minimum count is specified in the input record, zero (0) can be specified to indicate a multiple-value field containing no values.

If the number of values is the same for each record, this number may be specified with the field definition statement for the multiple-value field. In this case, the count byte in the input record must be omitted. If the record definitions are from an existing FDT or if the input data is from an earlier DECOMPRESS operation, the data already contains the length values; the count must not be specified in the field definition statements.

The count you specify may be changed by ADACMP if the NU option is specified for the field. ADACMP suppresses null values, and changes the count field accordingly. This is true whether you specify the value count before each series of values, or in the field definition statement. Refer to the section **Field/Group Definition/Multiple-Value Field (MU)** on page 73.

Example 1: Multiple-Value Field Count with Varying Number of Occurrences

Field Definition:

```
ADACMP FNDEF='01,MF,5,A,MU,NU'
```

Each record contains a different number of values for MF, and the count comes before each series of occurrences.

	Before ADACMP	After ADACMP
Input Record 1 (3 values)	MF count=3 AAAA BBBB CCCC	MF count=3 AAAA BBBB CCCC
Input Record 2 (2 values)	MF count=2 AAAA BBBB	MF count=2 AAAA BBBB
Input Record 3 (3 values)	MF count=3 AAAA bbbb CCCC	MF count=2 AAAA CCCC
Input Record 4 (no values)	MF count=0	MF count=0
Input Record 5 (1 value)	MF count=1 bbbb	MF count=0

Example 2: Multiple-Value Field Count with Same Number of Occurrences

Field Definition:

```
ADACMP FNDEF='01,MF,4,A,MU(3),NU'
```

Each record contains 3 values for MF, as specified in the field definition statement.

	Before ADACMP	After ADACMP
Input Record 1		MF count=3
	AAAA	AAAA
	BBBB	BBBB
	CCCC	CCCC
Input Record 2		MF count=2
	AAAA	AAAA
	BBBB	BBBB
	bbbb	
Input Record 3		MF count=2
	AAAA	AAAA
	bbbb	CCCC
	CCCC	
Input Record 4		MF count=0
	bbbb	
	bbbb	
	bbbb	

Periodic Group Count

Each periodic group must specify a count of field iterations (occurrences) in the record. The count is specified either within the field definition statement for all records, or as a one-byte binary value before each occurrence group in every record. If the count is in the field definition statement, the count byte must be omitted from the input records. When specified in the field definition statement, the minimum count allowed is 1, and the maximum number of periodic group occurrences allowed is 99 (or 191, if the MAXPE191 parameter is specified). When the minimum count is specified in the record, the value can be zero (0) for a periodic group with no occurrences.

The occurrence count provided may be modified by ADACMP if all the fields contained in the periodic group are defined with the NU option. If all the fields within a given occurrence contain null values and there are no following occurrences that contain non-null values, the occurrence will be suppressed and the periodic group occurrence count will be adjusted accordingly.

Example 1: Periodic Group Count with Varying Number of Occurrences

Field Definitions:

```
ADACMP COMPRESS MAXPE191,...
ADACMP FNDEF='01,GA,PE'
ADACMP FNDEF='02,A1,4,A,NU'
ADACMP FNDEF='02,A2,4,A,NU'
```

The input records contain a variable number of occurrences for GA (up to 191 occurrences are permitted). The count of occurrences comes before each occurrence group in the input records.

	Before ADACMP	**After ADACMP**
Input Record 1	GA count=2	GA count=2
	GA (1st occurrence) A1=AAAA A2=BBBB	 A1=AAAA A2=BBBB
	GA (2nd occurrence) A1=CCCC A2=DDDD	 A1=CCCC A2=DDDD
Input Record 2	GA count=1	GA count=0
	GA (1st occurrence) A1=bbbb A2=bbbb	 suppressed suppressed
Input Record 3	GA count=3	GA count=3
	GA (1st occurrence) A1=AAAA A2=bbbb	 A1=AAAA A2=suppressed
	GA (2nd occurrence) A1=BBBB A2=bbbb	 A1=BBBB A2=suppressed
	GA (3rd occurrence) A1=CCCC A2=bbbb	 A1=CCCC A2=suppressed
Input Record 4	GA count=0	GA count=0

Example 2: Periodic Group Count with Same Number of Occurrences

Field Definitions:

```
ADACMP FNDEF='01,GA,PE(3)'
ADACMP FNDEF='02,A1,4,A,NU'
ADACMP FNDEF='02,A2,4,A,NU'
```

All input records contain 3 occurrences for GA, as specified in the field definition statement.

	Before ADACMP	After ADACMP
Input Record 1		GA count=3
	GA (1st occurrence) A1=AAAA A2=bbbb	 A1=AAAA A2 suppressed
	GA (2nd occurrence) A1=BBBB A2=bbbb	 A1=BBBB A2 suppressed
	GA (3rd occurrence) A1=CCCC A2=bbbb	 A1=CCCC A2 suppressed
Input Record 2		GA count=2 (see note)
	GA (1st occurrence) A1=bbbb A2=bbbb	 A1=suppressed A2=suppressed
	GA (2nd occurrence) A1=BBBB A2=bbbb	 A1=BBBB A2=suppressed
	GA (3rd occurrence) A1=bbbb A2=bbbb	 A1=suppressed A2=suppressed
Input Record 3	All occurrences contain null values	GA count=0 All occurrences are suppressed

Note:
The first occurrence is included in the count since occurrences follow that contain non-null values. The third occurrence is not included in the count since there are no non-null values in the occurrences that follow.

Example 3: Adding a a Field to a PE-Group

In the PE named AW, the field AY should be added:

Old FDT	New FDT
01 AA,8,A,DE,UQ	01 AA,8,A,DE,UQ
01 AW,PE	01 AW,PE
02 AX,8,U,NU	02 AX,8,U,NU
02 AT,8,U,NU	02 AT,8,U,NU
01,AZ,3,A,DE,MU,NU	02 AY,8,U,NU
	01,AZ,3,A,DE,MU,NU

Note:
All of the currently existing fields in the PE must be specified.

1. Determine the maximum occurrence of the PE (for example, a result of 2).
2. Decompress the file with the format parameter.
3. Decompress INFILE=xx,FORMAT='AA,AX1-2,AT1-2,AZ'
4. Compress again:
   ADACMP COMPRESS FILE=32
   ADACMP FORMAT='AA,AX1-2,AT1-2,AZ'
   ADACMP FNDEF='01,AA,8,A,DE,UQ'
   ADACMP FNDEF='01,AW,PE(2)'
   ADACMP FNDEF='02,AX,8,U,NU'
   ADACMP FNDEF='02,AT,8,U,NU'
   ADACMP FNDEF='02,AY,8,U,NU'
   ADACMP FNDEF='01,AZ,3,A,DE,MU,NU'

Variable-Length Field Size

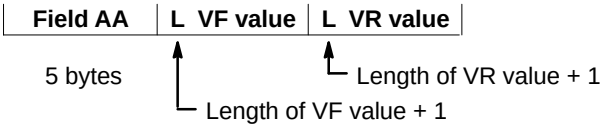
Each value of a variable-length field (length parameter not specified in the field definition) must be preceded by a one-byte binary count indicating the value length (including the length byte itself).

Example of Variable-Length Field Size

Field Definitions:

```
ADACMP FNDEF='01,AA,5,A,DE'
ADACMP FNDEF='01,VF,0,A'
ADACMP FNDEF='01,VR,0,A'
```

Input record:



Processing

Data Verification

ADACMP checks each field defined with format P (packed) or U (unpacked) to ensure that the field value is numeric and in the correct format. If a value is empty, the null characters must correspond to the format specified for the field (see the section **Representing SQL Null Characters** on page 78).

Alphanumeric (A)	blanks (hex ‘40’)
Binary (B)	binary zeros (hex ‘00’)
Fixed (F)	binary zeros (hex ‘00’)
Floating Point (G)	binary zeros (hex ‘00’)
Packed (P)	decimal packed zeros with sign (hex ‘00’ followed by ‘0F’, ‘0C’, or ‘0D’ in the rightmost, low-order byte)
Unpacked (U)	decimal unpacked zeros with sign (hex ‘F0’ followed by ‘C0’ or ‘D0’ in the rightmost, low-order byte)

Any record that contains invalid data is written to the ADACMP error (DD/FEHL) dataset and is not written to the compressed dataset.

Data Compression

The value for each field is compressed (unless the FI option is specified) as follows:

- Trailing blanks are removed for fields defined with A format.
- Leading zeros are removed for numeric fields (fields defined with B, F, P or U format).
- If the field is defined with U (unpacked) format, the value is converted to packed (P) format.
- Trailing zeros in floating-point (G format) fields are removed.
- If the field is defined with the NU option and the value is a null value, a one-byte indicator is stored. Hexadecimal ‘C1’ indicates one empty field follows, ‘C2’ indicates that two empty fields follow, and so on, up to a maximum of 63 before the indicator byte is repeated. For SQL null value (NC option field) compression, see page 78.
- Empty fields located at the end of the record are not stored, and therefore not compressed.

Example of Data Compression

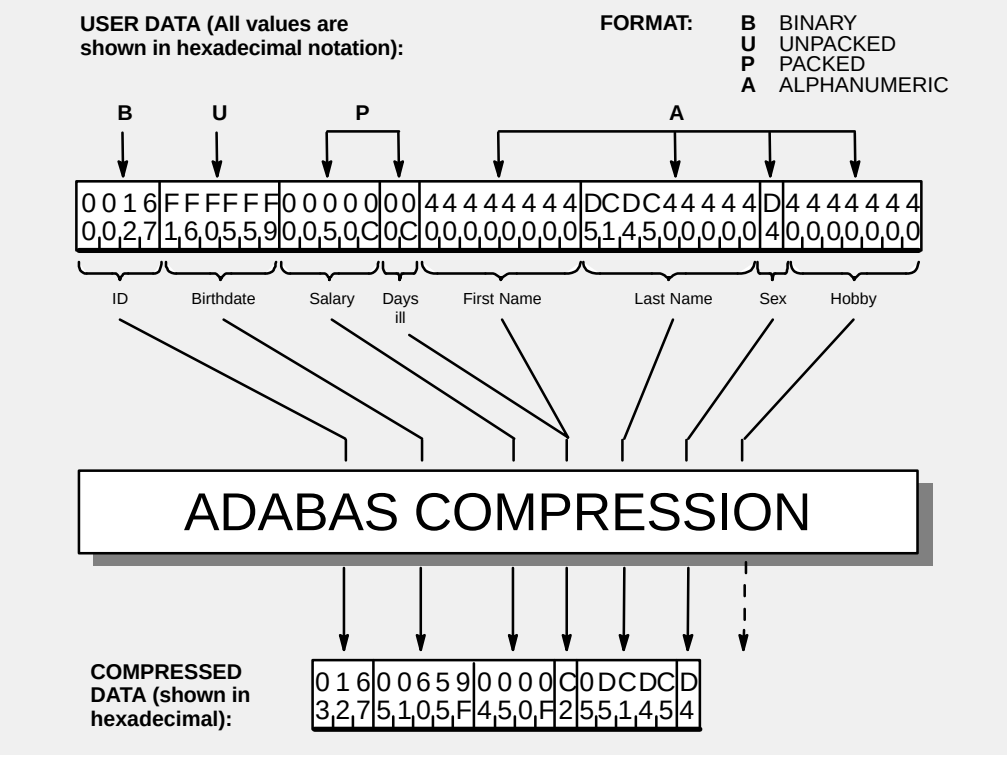


Figure 3-2: ADACMP Compression

Figure 3-2 shows how the following field definitions and corresponding values would be processed by ADACMP:

FNDEF='01,ID,4,B,DE'
FNDEF='01,BD,6,U,DE,NU'
FNDEF='01,SA,5,P'
FNDEF='01,DI,2,P,NU'
FNDEF='01,FN,9,A,NU'
FNDEF='01,LN,10,A,NU'
FNDEF='01,SE,1,A,FI'
FNDEF='01,HO,7,A,NU'

COMPRESS Function Output

Compressed Data Records

The data records that ADACMP has processed, edited, and compressed are written out together with the file definition information to a sequential dataset with the “variable blocked” record format. This dataset may be used as input to the ADALOD utility. The output of several ADACMP executions may also be used as input to ADALOD.

If the output dataset contains no records (no records provided on the input dataset or all records rejected), the output may still be used as input to the ADALOD utility. In this case, you must ensure that the amount of Associator space allocated to the file is sufficient since an accurate estimate cannot be made by the ADALOD utility without a representative sample of input record values (see the ADALOD utility for additional information).

Rejected Data Records

Any records rejected during ADACMP editing are written to the DD/FEHL error dataset. The records are output in variable blocked format and have the following structure:

Bytes	Description
0–1	Record length in binary format
2–3	Set to zero (X‘0000’)
4–5	Field name as stored in FDT
6–7	Offset from beginning of input record to error value
8–11	Input record sequence number (the first input record is “1”)
12	PE index (if applicable)
13	Adabas response code (in hexadecimal)*
14–15	(reserved; set to zeros)
16	DD/EBAND input record

***** Additionally the following response codes may occur:

X‘E7’(231) Input record too short (COMPRESS)
X‘E8’(232) Output record length error (COMPRESS)

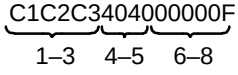
Only the first incorrect field within a record is detected and referenced. If there are other errors, they are not detected until subsequent runs are made.

Example of Rejected Data Records

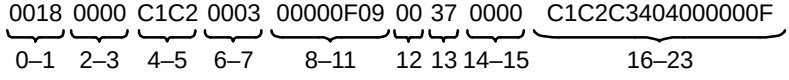
Field Definitions:

```
ADACMP FNDEF='01,AA,3,A,DE'
ADACMP FNDEF='01,AB,2,U'
ADACMP FNDEF='01,AC,3,P,NU'
```

Input record values (shown in hexadecimal): ISN = 3849 (decimal)



Rejected record as output by ADACMP (shown in hexadecimal):



The error dataset may be printed using the standard print utility provided with the operating system in use at the user installation. OS/390 or z/OS users may use the IEBPTPCH utility. VSE/ESA users may use the DITTO program. RDW (record descriptor word, bytes 1-4) may or may not be present, depending on the print utility used.

ADACMP Report

ADACMP calculates the approximate amount of space (in both blocks and cylinders) required for Data Storage for the compressed records. This information is printed as a matrix which contains the required space for the different device types requested by the DEVICE parameter for various Data Storage padding factors between 5 and 30 percent.

The following is an example of ADACMP report output:

```
PARAMETERS:

ADACMP COMPRESS NUMREC=1000
ADACMP FNDEF='01,AA,8,B,DE'
ADACMP FNDEF='01,BA,6,A,NU'
ADACMP FNDEF='01,BB,8,P,NU'
ADACMP FNDEF='01,AD,1,A,FI'
ADACMP SUBDE='CA=BA(1,3)'
```

```
COMPRESS PROCESSING STATISTICS:

NUMBER OF RECORDS READ                1,000
NUMBER OF INCORRECT RECORDS               0
NUMBER OF COMPRESSED RECORDS          1,000

RAW DATA                             24,000 BYTES
COMPRESSED DATA                      16,656 BYTES
COMPRESSION RATE                       31.9 %
LARGEST COMPRESSED RECORD                20 BYTES

DATASTORAGE SPACE REQUIREMENTS:

I  DEVICE  I  PADDING I  BLOCKSIZE I          NUMBER OF     I
I          I  FACTOR  I    BYTES   I      BLOCKS      CYLS  I
I----------I----------I------------I------------------------I
I   3380   I          I    4,820   I                        I
I          I     5%   I    4,578   I           4         1  I
I          I    10%   I    4,337   I           4         1  I
I          I    15%   I    4,096   I           5         1  I
I          I    20%   I    3,856   I           5         1  I
I          I    25%   I    3,615   I           5         1  I
I          I    30%   I    3,373   I           5         1  I
I          I          I            I                        I
I----------I----------I------------I------------------------I


TEMP SPACE ESTIMATION:

I  DEVICE  I  BLOCKSIZE I          NUMBER OF     I
I          I    BYTES   I      BLOCKS      CYLS  I
I----------I------------I------------------------I
I   3380   I    7,476   I           5          1 I
I----------I------------I------------------------I

THE LARGEST DESCRIPTOR IS AA, IT WILL OCCUPY 1 TEMP BLOCKS

SORT SPACE ESTIMATION:

I DEVICE I BLOCKSIZE I   LWP              I      NR OF      I
I        I  (BYTES)  I  (BYTES)           I  BLOCKS    CYLS I
-------------------------------------------------------------
I  3380  I     7476  I   139264 (MINIMUM) I       2       1 I
I        I           I  1048576 (DEFAULT) I       2       1 I
I        I           I   139264 (OPTIMUM) I       2       1 I
I--------I-----------I--------------------I-----------------I
```

The compression rate is computed based on the real amount of data used as input to the compression routine. Fields skipped by a format element “nX” (used to fill a field with blanks) are not counted.

DECOMPRESS Function Output

The ADACMP DECOMPRESS function decompresses each record and then stores the record in a sequential dataset. The records are output in variable-length, blocked format. Each decompressed record is output either with or without the ISN option according to the format shown below:

length xx [**ISN**] data

—where

length	is a two-byte binary length of the data, + 8 (or +4 if the ISN parameter is not specified).
xx	is a two-byte field containing binary zeros.
ISN	is a four-byte binary ISN of the record.
data	is a decompressed data record.

The fields of the data record are provided in the order in which they appeared in the FDT when the file was unloaded. The standard length and format are in effect for each field.

If a field value exceeds the standard length, the value will be truncated to the standard length if the field is alphanumeric and the TRUNCATE parameter was specified; otherwise, ADACMP writes the record to the DD/FEHL error dataset (see the following section).

Any count bytes for multiple-value fields or periodic groups contained in the record are included in the decompressed data output. ADACMP generates a count of 1 if the MU field or PE group is empty. This makes it possible to use the output of the DECOMPRESS operation as the input to a subsequent COMPRESS operation.

Rejected Data Records

Data records rejected by the DECOMPRESS operation are written to the DD/FEHL error dataset. ADACMP rejects a record whenever a compressed field's size is greater than the default length held in the FDT, unless the TRUNCATE parameter is specified.

The records are output in variable blocked format, and have the following structure:

Bytes	Description
0–1	Record length in binary format (see note 2 below)
2–3	Set to zero; that is, X'0000' (see note 2 below)
4–5	Field name as stored in FDT
6–7	Offset from beginning of input record to error value
8–11	ISN in binary format
12	PE index (if applicable)
13	Adabas response code (in hexadecimal)*
14–15	(reserved; set to zeros)
16	DD/EBAND input record

***** Additionally the following response codes may occur:

X'E7'(231) Input record too short (DECOMPRESS)
X'E8'(232) Output record length error (DECOMPRESS)

Notes:

1. *Only the first incorrect field within a record is detected and referenced in DD/FEHL. Other errors within the record are not detected or recorded.*
2. *Bytes 0–1 and 2–3 are not visible when the output record is viewed from an editor. However, the bytes are provided when the record is accessed from an application program.*

Restart Considerations

ADACMP has no restart capability. An interrupted ADACMP execution must be reexecuted from the beginning.

User Exit 6

A user-written routine called user exit 6 can be used for editing during ADACMP COMPRESS processing. The routine may be written in Assembler or COBOL. It must be assembled or compiled and then linked into the Adabas load library (or any library concatenated with it).

User exit 6 is invoked by specifying

ADARUN UEX6=program

—where “program” is the routine name in the load library.

See the *Adabas DBA Reference Manual*, chapter **User Exits** for specific information about the user exit 6 structure and parameters.

COMPRESS : Create an Adabas File

ADACMP COMPRESS [**CODE=**cipher-code]
[**DEVICE=**{ device-type-list | ADARUN-device }]
[**FACODE=**file-alpha-EBCDIC-key]
[**FDT=**file-number]
[**FILE=** { file-number|**0** }]
[**FWCODE=**file-wide-key]
[**FUWCODE=**{wide-key | UWCODE-definition }]
[**FORMAT=**format]
[**MAXPE191**]
[**NOUSERABEND**]
[**NUMREC=**number-of-records]
[**PASSWORD=’**password**’**]
[**RECFM=** { **F**|**FB**|**V**|**VB**|**U** }]
[**,LRECL=**record-length]
[{ **USERISN** | **MINISM=***{* start-isn|**1** } }]
[**UACODE=**userdata-alpha-key]
[**UWCODE=**{userdata-wide-key | FWCODE-definition }]
[**UARC=**{userdata-architecture-key | **2**}]
. . . FIELD DEFINITION STATEMENTS
FNDEF=’field-definition**’**
[**COLDE=’**collation-descriptor-definition**’**]
[**HYPDE=’**hyperdescriptor-definition**’**]
[**PHONDE=’**phoneticdescriptor-definition**’**]
[**SUBDE=’**subdescriptor-definition**’**]
[**SUBFN=’**subfield-definition**’**]
[**SUPDE=’**superdescriptor-definition**’**]
[**SUPFN=’**superfield-definition**’**]

Optional Parameters and Subparameters

CODE : Cipher Code

If the data is to be loaded into the database in ciphered form, the cipher code must be specified with this parameter. See the *Adabas Security Manual* for additional information on the use of ciphering.

DEVICE : Device Type

ADACMP calculates and displays a report of this run's space requirements for each specified device type. If DEVICE= is not specified, the default is the ADARUN device type.

FACODE : Alphanumeric Field Encoding

FACODE must be specified if you want to define UES file encoding for alphanumeric fields in the file. The alphanumeric encoding must belong to the EBCDIC encoding family; that is, the space character is X'40'.

FDT : Use Existing Adabas Field Definition Table

An existing Adabas FDT is to be used. The FDT may be that of an existing file or a file that has been deleted with the KEEPFDT option of the ADADBS utility.

The input data must be consistent with the structure as defined in the specified FDT, unless the FORMAT parameter is used. When the FDT defines multiple-value fields or periodic groups, length values must be defined or already included in the FDT; refer to the sections **Multiple-Value Field Count** on page 45 and **Periodic Group Field Count** on page 47.

If the FDT parameter is used, any field definitions specified will be ignored.

FILE : File Number

If the FDT contains a hyperdescriptor, this parameter must be specified. The specified file number becomes input for the related hyperexit. For more information about hyperexits, refer to the *Adabas DBA Reference Manual*.

User exit 6 is always supplied with this file number. If FILE is not specified, a value of zero is assumed.

FORMAT : Input Record Format Definition

Use this parameter to provide a format definition that indicates the location, format, and length of fields in the input record. The format provided must follow the rules for format buffer entries for update commands as described in the *Adabas Command Reference Manual*.

Conversion rules are those described for Adabas update commands in the *Adabas Command Reference Manual*. For conversion of SQL null (NC option) field values, see page 79. If a field is omitted in the FORMAT parameter, that field is assigned no value.

If the FORMAT parameter is omitted, the input record is processed in the order of the field definition statements provided or, if the FDT parameter is used, according to an existing Adabas field definition table.

FUWCODE : Wide-Character Field Default User Encoding

FUWCODE defines the default user encoding for wide-character fields for the file when loaded in the database. If this parameter is omitted, the encoding is taken from the UWCODE definition of the database.

FWCODE : Wide-Character Field Encoding

If fields with format W (wide-character) exist in the compressed file, you **must** specify FWCODE to define the file encoding for them.

FWCODE also determines the maximum byte length of the wide-character field.

LRECL : Input Record Length (VSE Only)

If RECFM=F or RECFM=FB is specified, this parameter must also be specified to provide the record length (in bytes) of the input data; otherwise, do not specify LRECL.

For z/OS or OS/390, the record length is taken from the input dataset label or DD statement.

For BS2000, the record length is taken from the catalog entry or /FILE statement.

MAXPE191 : Enable Periodic Group Count Up to 191

Periodic groups can have up to 191 occurrences. The limit of 191 is allowed by the nucleus without further specification; however, to compress records with more than 99 periodic group occurrences, the parameter MAXPE191 must be specified.

Note:
This option is not compatible with Adabas 5.2 releases; therefore, backward conversion to Adabas 5.2 is not possible once records with more than 99 PE group occurrences have been loaded.

MINISN : Starting ISN

For automatic ISN assignment, MINISN defines the lowest ISN to be used. If MINISN is not specified, the default is 1. If USERISN is specified, MINISN cannot be specified.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NUMREC : Number of Records to Be Processed

Specifies the number of input records to be processed. If this parameter is omitted, all input records contained on the input dataset are processed.

Software AG recommends using this parameter for the initial ADACMP execution if a large number of records are contained on the input dataset. This avoids unneeded processing of all records when a field definition error or invalid input data results in a large number of rejected records. This parameter is also useful for creating small files for test purposes.

Setting NUMREC to zero (0) prevents the input dataset from being opened.

PASSWORD : Password for FDT File

If the FDT parameter is specified and the file is password-protected, this parameter must be used to provide a valid password for that file.

RECFM : Input Record Format (VSE Only)

You **must** specify the input record format with this parameter as follows:

F	fixed length, unblocked (requires that you also specify the LRECL parameter)
FB	fixed length, blocked (requires that you also specify the LRECL parameter)
V	variable length, unblocked
VB	variable length, blocked
U	undefined

Under z/OS or OS/390, the record format is taken from the input dataset label or DD statement.

Under BS2000, the record format is taken from the catalog entry or FILE statement.

UACODE : User Encoding for Input Alphanumeric Fields

UACODE defines the user encoding of the sequential input of alphanumeric fields. If you specify UACODE, you **must** also specify FACODE.

UARC : Architecture for Input Uncompressed User Data

The UARC parameter specifies the architecture of the sequential input of the uncompressed user data. The “userdata-architecture-key“ is an integer which is of the sum of the following numbers:

byte order	b=0	high-order byte first
	b=1	low-order byte first
encoding family	e=0	ASCII encoding family
	e=2	EBCDIC encoding family (default)
floating-point format	f=0	IBM370 floating-point format
	f=4	VAX floating-point format
	f=8	IEEE floating-point format

The default is ARC = b + e + f = 2; that is, high-order byte first; EBCDIC encoding family; and IBM370 floating-point format (b=0; e=2; f=0).

User data from an Intel386 PC provides the example: b=1; e=0; f=8; or ARC=9.

USERISN : User ISN Assignment

The ISN for each record is to be provided by the user. If this parameter is omitted, the ISN for each record is assigned by Adabas.

If USERISN is specified, the user must provide the ISN to be assigned to each record as a four-byte binary number immediately preceding each data record. If the MINISN parameter is specified, USERISN cannot be specified.

The format for fixed or undefined length input records with user-defined ISNs is

```
userisn/data
```

The format for variable-length input records with user-defined ISNs is

```
length/xx/userisn/data
```

—where

length	is a two-byte binary physical record length (length of record data, plus 8 bytes).
xx	is a two-byte field containing binary zeros.
userisn	is a four-byte binary ISN to be assigned to the record.
data	is input record data.

ISNs may be assigned in any order, must be unique (for the file), and must not exceed the MAXISN setting specified for the file (see the ADALOD utility).

ADACMP does not check for unique ISNs or for ISNs that exceed MAXISN. These checks are performed by the ADALOD utility.

UWCODE : User Encoding for Input Wide-Character Fields

UWCODE defines the user encoding of the sequential input of wide-character fields. If you specify UWCODE, you **must** also specify FWCODE.

For user input, all wide-character fields are encoded in the same code page. It is not possible to select different encodings for different fields in the same ADACMP run.

Essential Data Definition Syntax

The field definitions provided as input to ADACMP are used to

- provide the length and format of each field contained in the input record. This enables ADACMP to determine the correct field length and format during editing and compression;
- create the field definition table (FDT) for the file. This table is used by Adabas during the execution of Adabas commands to determine the logical structure and characteristics of any given field (or group) in the file.

The following syntax must be followed when entering field definitions. A minimum of one and a maximum of 926 definitions may be specified.

Field	**FNDEF=’**level**,** name**,** length**,** format [{,option}...] **’**
Group	**FNDEF=’**level**,** name [**,PE** [**(n)**]]**’**
Collation descriptor	**COLDE=’**number**,**name [**,UQ** [,**XI**]]=parentfield**’**
Hyperdescriptor	**HYPDE=’**number**,**name**,**length**,**format [{,option}...]= {parentfield}**,...’**
Phonetic descriptor	**PHONDE=’**name(field)**’**
Subdescriptor	**SUBDE=’**name [**,UQ** [,**XI**]] = parentfield (begin**,**end)**’**
Subfield	**SUBFN=’**name = parentfield (begin**,**end)**’**
Superdescriptor	**SUPDE=’**name[**,UQ** [,**XI**]] = {parentfield (begin**,**end)} **,...’**
Superfield	**SUPFN=’**name = {parentfield (begin**,**end)} **,...’**

User comments may be entered to the right of each definition. At least one blank must be present between a definition and any user comments.

FNDEF : Field/Group Definition

This parameter is used to specify an Adabas field (data) definition. The syntax used in constructing field definition entries is

FNDEF= ‘level**,** name **[,** length**,** format**]** **[{,**option**}...]**’

Level number and name are required. Any number of spaces may be inserted between definition entries.

level

The level number is a one- or two-digit number in the range 01–07 (the leading zero is optional) used in conjunction with field grouping. Fields assigned a level number of 02 or greater are considered to be a part of the immediately preceding group which has been assigned a lower level number.

The definition of a group enables reference to a series of fields (may also be only 1 field) by using the group name. This provides a convenient and efficient method of referencing a series of consecutive fields.

Level numbers 01–06 may be used to define a group. A group may consist of other groups. When assigning the level numbers for nested groups, no level numbers may be skipped.

```
FNDEF='01,GA'       group
FNDEF='02,A1,...'           elementary or multiple-value field
FNDEF='02,A2,...'           elementary or multiple-value field
FNDEF='01,GB'       group
FNDEF='02,B1,...'           elementary or multiple-value field
FNDEF='02,GC'       group (nested)
FNDEF='03,C1,...'           elementary or multiple-value field
FNDEF='03,C2,...'           elementary or multiple-value field
```

Fields A1 and A2 are in group GA. Field B1 and group GC (consisting of fields C1 and C2) are in group GB.

name

The name to be assigned to the field (or group).

Names must be unique within a file. The name must be two characters long: the first character must be alphabetic; the second character can be either alphabetic or numeric. No special characters are permitted.

The values E0-E9 are reserved as edit masks and may not be used.

Valid Names	Invalid Names
AA	A (not two characters)
B4	E3 (edit mask)
S3	F* (special character)
WM	6M (first character not alphabetic)

length

The length of the field (expressed in bytes). The length value is used to

- indicate to ADACMP the length of the field as it appears in each input record; and
- define the standard (default) length to be used by Adabas during command processing.

The standard length specified is entered in the FDT and is used when the field is read/updated unless the user specifies a length override.

The maximum field lengths that may be specified depend on the "format" value:

Format	Maximum Length
Alphanumeric (A)	253 bytes
Binary (B)	126 bytes
Fixed Point (F)	4 bytes (always exactly 2 or 4 bytes)
Floating Point (G)	8 bytes (always exactly 4 or 8 bytes)
Packed Decimal (P)	15 bytes
Unpacked Decimal (U)	29 bytes
Wide-character (W)	253 bytes*

***** Depending on the FWCODE attribute value, the maximum byte length of the W field may be less than 253. For example, if the default value of FWCODE is used (that is, Unicode), the maximum length is 252 (2 bytes per character).

Standard length may not be specified with a group name.

Standard length does not limit the size of any given field value unless the FI option is used — see page 71. A read or update command may override the standard field length, up to the maximum length permitted for that format.

If standard length is zero for a field, the field is assumed to be a variable-length field. Variable-length fields have no standard (default) length. A length override for fixed-point (F) fields can specify a length of two or four bytes only; for floating-point (G) fields, the override can specify four or eight bytes only.

If a variable-length field is referenced without a length override during an Adabas command, the value in the field will be returned preceded by a one-byte binary length field (including the length byte itself). This length value must be specified when the field is updated, and also in the input records that are to be processed by ADACMP. If the field is defined with the long alpha (LA) option, the value is preceded by a two-byte binary length field (including the two length bytes).

format

The standard format of the field (expressed as a one-character code):

A	Alphanumeric (left-justified)
B	Binary (right-justified, unsigned/positive)
F	Fixed point (right-justified, signed, two's complement notation)
G	Floating point (normalized form, signed)
P	Packed decimal (right-justified, signed)
U	Unpacked decimal (right-justified, signed)
W	Wide character (left-justified)

The standard format is used to

- indicate to ADACMP the format of the field as it appears in each input record; and
- define the standard (default) format to be used by Adabas during command processing. The standard format specified is entered in the FDT and is used when the field is read/updated unless the user specifies a format override.

Standard format must be specified for a field. It may not be specified with a group name. When the group is read (written), the fields within the group are always returned (must be provided) according to the standard format of each individual field. The format specified determines the type of compression to be performed on the field.

A fixed-point field is either two or four bytes long. A positive value is in normal form, and a negative value in two's complement form.

A field defined with floating-point format may be either four bytes (single precision) or eight bytes (double precision) long. Conversion of a value of a field defined as floating point to another format is supported.

If a binary field is to be defined as a descriptor, and the field may contain both positive and negative numbers, "F" format should be used instead of "B" format because "B" format assumes that all values are unsigned (positive).

Like an alphanumeric field, a wide-character field may be a standard length in bytes defined in the FDT, or variable length. Any non-variable format override for a wide-character field must be compatible with the user encoding; for example, a user encoding in Unicode requires an **even** length. Format conversion from numbers (U, P, B, F, G) to wide-character format is not allowed.

Data Definition Field/Group Options

Options are specified by the two-character codes. These codes may be specified in any order, separated by a comma.

Code	Option	Page
DE	Field is to be a descriptor (key).	70
FI	Field is to have a fixed storage length; values are stored without an internal length byte, are not compressed, and cannot be longer than the defined field length.	71
LA	This A or W format variable-length field may contain a value up to 16,381 bytes long.	72
MU	Field may contain up to 191 values in a single record.	73
NC	Field may contain a null value that satisfies the SQL interpretation of a field having no value; that is, the field's value is not defined (not counted).	79
NU	Null values occurring in the field are to be suppressed.	75
NV	This A or W format field is to be processed in the record buffer without being converted.	76
PE	Group field is to be followed by a periodic group definition that may occur up to 191 times in a given record.	76
NN	Field defined with NC option must always have a value defined; it cannot contain an SQL null (not null).	81
UQ	Field is to be a unique descriptor; that is, for each record in the file, the descriptor must have a different value.	77
XI	For this field, the index (occurrence) number is excluded from the UQ option set for a PE.	77

DE : Descriptor

DE indicates that the field is to be a descriptor (key). Entries will be made in the Associator inverted list for the field, enabling the field to be used in a search expression, as a sort key in a FIND command, to control logical sequential reading, or as the basis for file coupling.

The descriptor option should be used judiciously, particularly if the file is large and the field that is being considered as a descriptor is updated frequently.

Although the definition of a descriptor field is independent of the record structure, note that if a descriptor field is not ordered first in a record and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to reorder the field first for each record of the file.

FI : Fixed Storage

FI indicates that the field is to have a fixed storage length. Values in the field are stored without an internal length byte, are not compressed, and cannot be longer than the defined field length.

The FI option is recommended for fields with a length of one or two bytes that have a low probability of containing a null value (personnel number, gender, etc.) and for fields containing values that cannot be compressed.

The FI option is not recommended for multiple-value fields, or for fields within a periodic group. Any null values for such fields are not suppressed (or compressed), which can waste disk storage space and increase processing time.

The FI option **cannot** be specified for

- U-format fields;
- NC, NN, or NU option fields;
- variable-length fields defined with a length of zero (0) in the FNDEF statement;
- a descriptor within a periodic (PE) group.

A field defined with the FI option **cannot** be updated with a value that exceeds the standard length of the field.

Example of FI usage:

	Definition	**User Data**	**Internal Representation**
Without FI Option	FNDEF=’01,AA,3,P’	33104C 00003C	0433104F (4 bytes) 023F (2 bytes)
With FI Option	FNDEF=’01,AA,3,P,FI’	33104C 00003C	33104F (3 bytes) 00003F (3 bytes)

LA : Long Alpha Option

The LA (long alphanumeric) option can be specified for variable-length alphanumeric and wide format fields; i.e., A or W format fields having a length of zero in the field definition (FNDEF). With the LA option, such a field can contain a value up to 16,381 bytes long.

An alpha or wide field with the LA option is compressed in the same way as an alpha or wide field without the option. The maximum length that a field with LA option can actually have is restricted by the block size where the compressed record is stored.

When a field with LA option is updated or read, its value is either specified or returned in the record buffer, preceded by a two-byte length value that is inclusive (field length, plus two).

A field with LA option

- can also have the NU, NC/NN, or MU option;
- can be a member of a PE group;
- cannot have the FI option;
- cannot be a descriptor field;
- cannot be a parent of a sub-/superfield, sub-/superdescriptor, hyperdescriptor, or phonetic descriptor; and
- cannot be specified in the search buffer, or response code 61 occurs.

For more information, see the *Adabas Command Reference Manual* section **Specifying a Field with LA (Long Alpha) Option** in the chapter 2 discussion of the record buffer.

Example of LA usage:

	Definition	**User Data**	**Internal Representation**
Without LA Option	FNDEF=’01,BA,0,A’	X’06’,C’HELLO’ —	X’06C8C5D3D3D6’ (1-byte length) —
With LA Option	FNDEF=’01,BA,0,A,LA’	X’0007’,C’HELLO’ X’07D2’,C’ ... (2000 data bytes) ...’	X’06C8C5D3D3D6’ (1-byte length) X’87D2 ... (2000 data bytes) ... ’

MU : Multiple-Value Field

MU indicates that the field may contain more than one value in a single record. The actual number of values present in each record may vary from 0 to 191, although at least one value (even if null) must be present in each record input to ADACMP.

The values are stored according to the other options specified for the field. The first value is preceded by a count field that indicates the number of values currently present for the field. The number of values that are stored is equal to the number of values provided in the ADACMP input record, plus any values added during later updating of the field, less any values suppressed (this applies only if the field is defined with the NU option).

If the number of values contained in each record input to ADACMP is constant, the number can be specified in the MU definition statement in the form MU(n), where “n” equals the number of values present in each input record. For example:

```
FNDEF='01,AA,5,A,MU(3)'
```

—indicates that three values of the multiple-value field AA are present in each input record. Specifying a value of zero (0) indicates that no values are present for the multiple-value field in the input record.

If the number of values is not constant for all input records, a one-byte binary count field must precede the first value in each input record to indicate the number of values present in that record (see also the section **Input Data Requirements** on page 45).

If the FDT is provided (see the FDT parameter description on page 61), the field count must be contained as a one-byte binary value in each input record.

If the input records were created using the DECOMPRESS function, all required count fields are already contained in the input record. In this case, the count must not be specified in the field definition statement.

All values provided during input or updating will be compressed (unless the FI option has also been specified). Care should be taken when using the FI and MU options together since a large amount of disk storage may be wasted if a large number of compressible values are present.

If the NU option is specified with the MU option, null values are both logically and physically suppressed. The positional relationship of all values (including null values) is maintained in MU occurrences, unless the occurrences are defined with the NU option. If a large number of null values are present in an MU field group, the NU option can reduce the disk storage requirements for the field but should not be used if the relative positions of the values must be maintained.

The NC (or NC/NN) option **cannot** be specified for an MU field.

Example of MU usage **with** NU:

FNDEF='01,AA,5,A,MU,NU'

The original content where "L" is the length of the "value" is

- after file loading:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

- after update of value B to null value:

2	L value A	L value C
count	AA1	AA2

Example of MU usage **without** NU:

FNDEF='01,AA,5,A,MU'

The original content where "L" is the length of the "value" is

- after file loading:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

- after update of value B to null value:

3	L value A	L value B	L value C
count	AA1	AA2	AA3

NU : Null Value Suppression

NU suppresses null values occurring in the field.

Normal compression (NU or FI not specified) represents a null value with two bytes (the first for the value length, and the second for the value itself, in this case a null). Null value suppression represents an empty field with a one-byte “empty field” indicator. The null value itself is not stored.

A series of consecutive fields containing null values and specifying the NU option is represented by a one-byte “empty field” (binary 11nnnnnn) indicator, where “nnnnnn” is the number of the fields’ successive bytes containing null values, up to a total of 63. For this reason, fields defined with the NU option should be grouped together whenever possible.

If the NU option is specified for a descriptor, any null values for the descriptor are not stored in the inverted list. Therefore, a find command in which this descriptor is used and for which a null value is used as the search value will always result in no records selected, even though there may be records in Data Storage that contain a null value for the descriptor. If a descriptor defined with the NU option is used to control a logical sequence in a read logical sequence (L3/L6) command, those records that contain a null value for the descriptor will not be read.

Descriptors to be used as a basis for file coupling and for which a large number of null values exist should be specified with the NU option to reduce the total size of the coupling lists.

The NU option cannot be specified for fields defined with the combined NC/NN options or with the FI option.

Example of NU usage:

	Definition	**User Data**	**Internal Representation**
Normal Compression	FNDEF=’01,AA,2,B’	0000	0200 (2 bytes)
With FI Option	FNDEF=’01,AA,2,B,FI’	0000	0000 (2 bytes)
With NU Option	FNDEF=’01,AA,2,B,NU’	0000	C1 (1 byte)*

***** C1 indicates 1 empty field.

NV : No Conversion

The “do not convert” option for alphanumeric (A) or wide-character (W) format fields specifies that the field is to be processed in the record buffer without being converted.

Fields with the NV option are not converted to or from the user: the field has the characteristics of the file encoding; that is, the default blank

- for A fields, is always the EBCDIC blank (X’40’); and
- for W fields, is always the blank in the file encoding for W format.

The NV option is used for fields containing data that cannot be converted meaningfully or should not be converted because the application expects the data exactly as it is stored.

The field length for NV fields is byte-swapped if the user architecture is byte-swapped.

For NV fields, “A” format cannot be converted to “W” format and vice versa.

PE : Periodic Group

PE indicates that a periodic group is to be defined. A periodic group

- may comprise one or more fields. A maximum of 254 elementary fields may be specified. Descriptors and/or multiple value fields and other groups may be specified, but a periodic group may not contain another periodic group.
- may occur from 0 to 99 (or 191, if the ADACMP MAXPE191 parameter is specified) times within a given record, although at least one occurrence (even if it contains all null values) must be present in each ADACMP input record.
- must be defined at the 01 level. All fields in the periodic group must immediately follow and must be defined at level 02 or higher (in increments of 1 to a maximum of 7). The next 01 level definition indicates the end of the current periodic group.
- may only be specified with a group name. Length and format parameters may not be specified with the group name.

Following are two examples of period group definition:

Periodic Group “GA”:

```
FNDEF=’01,GA,PE’
FNDEF=’02,A1,6,A,NU’
FNDEF=’02,A2,2,B,NU’
FNDEF=’02,A3,4,P,NU’
```

Periodic Group “GB”:

```
FNDEF=’01,GB,PE(3)’
FNDEF=’02,B1,4,A,DE,NU’
FNDEF=’02,B2,5,A,MU(2),NU’
FNDEF=’02,B3’
FNDEF=’03,B4,20,A,NU’
FNDEF=’03,B5,7,U,NU’
```

UQ : Unique Descriptor

UQ indicates that the field is to be a unique descriptor. A unique descriptor must contain a different value for each record in the file. In FNDEF statements, the UQ option can only be specified if the DE option is also specified. The UQ option can also be used in SUBDE, SUPDE, and HYPDE statements.

The UQ option **must** be specified if the field is to be used as an ADAM descriptor (see the ADAMER utility).

ADACMP does not check for unique values; this is done by the ADALOD utility, or by the ADAINV utility when executing the INVERT function. If a non-unique value is detected during file loading, ADALOD terminates with an error message.

Because ADAINV and ADALOD must execute separately for each file in an expanded file chain, they cannot check for uniqueness across the chain.

However, Adabas does checks the value of unique descriptors across an expanded file chain. If the value being added (N1/N2) or updated (A1) is not unique across all files within the chain, response code 198 is returned.

XI : Exclude Instance Number

By default, the occurrence number of fields within periodic groups (PE) defined as unique descriptors (UQ) is included as part of the descriptor value. This means that the same field value can occur in different periodic group occurrences in different records.

The XI option is used to exclude the occurrence number from the descriptor value for the purpose of determining the the value’s uniqueness. If the XI option is set, any field value can occur at most once over all occurrences of the PE field in all records.

Representing SQL Null Values

Adabas includes two data definition options, NC and NN, to provide SQL-compatible null representation for Software AG's mainframe Adabas SQL Server (ESQ) and other Structured Query Language (SQL) database query languages.

The NC and NN options **cannot** be applied to fields defined

- with Adabas null suppression (NU)
- with fixed-point data type (FI)
- with multiple-values (MU)
- within a periodic group (PE)
- as group fields

In addition, the NN option can only be specified for a field that specifies the NC option.

A parent field for sub-/superfields or sub-/superdescriptors can specify the NC option. However, parent fields for a single superfield or descriptor cannot use a mix of NU and NC fields. If any parent field is NC, no other parent field can be an NU field, and vice versa.

Examples:

A correct ADACMP COMPRESS FNDEF statement for defining the field AA and assigning the NC and NN option:

```
ADACMP  FNDEF='01,AA,4,A,NN,NC,DE'
```

Incorrect uses of the NC/NN option that would result in an ADACMP utility ERROR–127:

Incorrect Example	Reason
ADACMP FNDEF='01,AA,4,A,NC,NU'	NU and NC options are not compatible
ADACMP FNDEF='01,AB,4,A,NC,FI'	NC and FI options are not compatible
ADACMP FNDEF='01,PG,PE' ADACMP FNDEF='02,P1,4,A,NC'	NC option within a PE group is not allowed

NC : SQL Null Value Option

Without the NC (not counted) option, a null value is either zero or blank depending on the field's format.

With the NC option, zeros or blanks specified in the record buffer are interpreted according to the "null indicator" value: either as true zeros or blanks (that is, as "significant" nulls) or as undefined values (that is, as true SQL or "insignificant" nulls).

If the field defined with the NC option has no value specified in the record buffer, the field value is always treated as an SQL null.

When interpreted as a true SQL null, the null value satisfies the SQL interpretation of a field having no value. This means that no field value has been entered; that is, the field's value is not defined.

The null indicator value is thus responsible for the internal Adabas representation of the null. For more information, see the following section **Null Indicator Value** and the section **Search Buffer Syntax** in the *Adabas Command Reference Manual*.

The following rules apply when compressing or decompressing records containing NC fields:

1. If the FORMAT parameter is specified, ADACMP behaves in the same way the nucleus does for update-type commands. See the *Adabas Command Reference Manual*.

2. If the FORMAT parameter is **not** specified

   - for **compression**

   Only the value of the NC field is placed in the input record; the two null value indicator bytes must be omitted. The value is compressed as if the null value indicator bytes were set to zero. It is not possible to assign a null value to an NC field using this method.

Example:

Field Definition Table (FDT) definition:	**FNDEF='01,AA,4,A,NC'**
Input record contents:	**MIKE**

- for **decompression**

  If the value of an NC field is **not significant**, the record is written to DDFEHL (or FEHL) with response code 55.

  If the value of an NC field is **significant**, the value is decompressed as usual. There are no null indicator bytes.

Example:

Field Definition Table (FDT) definition:	**FNDEF='01,AA,4,A,NC'**
Output record contents:	**MIKE**

Null Indicator Value

The null indicator value is always two bytes long and has fixed-point format, regardless of the data format. It is specified in the record buffer when a field value is added or changed; it is returned in the record buffer when the field value is read.

For an update (Ax) or add (Nx) command, the null indicator value must be set in the record buffer position that corresponds to the field's designation in the format buffer. The setting must be one of the following:

Hex Value	Indicates that . . .
FFFF	the field's value is set to "undefined", an insignificant null; the differences between no value, binary zeros, or blanks for the field in the record buffer are ignored; all are interpreted equally as "no value".
0000	no value, binary zeros, or blanks for the field in the record buffer are interpreted as significant null values.

For a read (Lx) or find with read (Sx with format buffer entry) command, your program must examine the null indicator value (if any) returned in the record buffer position corresponding to the field's position in the format buffer. The null indicator value is one of the following values, indicating the meaning of the actual value that the selected field contains:

Hex Value	Indicates that . . .
FFFF	a zero or blank in the field is not significant.
0000	a zero or blank in the field is a significant value; that is, a true zero or blank.
xxxx	the field is truncated. The null indicator value contains the length (xxxx) of the entire value as stored in the database record.

Example:

The field definition of a null represented in a two-byte Adabas binary field AA defined with the NC option is

01,AA,2,B,NC

For a . . .	Null Indicator Value (Record Buffer)	Data	Adabas Internal Representation
non-zero value	0 (binary value is significant)	0005	0205
blank	0 (binary null is significant)	0000 (zero)	0200
null	FFFF (binary null is not significant)	(not relevant)	C1

NN : SQL Not Null Option

The NN ("not null" or "null value not allowed") option may only be specified when the NC option is also specified for a data field. The NN option indicates that an NC field must always have a value (including zero or blank) defined; it cannot contain "no value".

The NN option ensures that the field will not be left undefined when a record is added or updated; a significant value must always be set in the field. Otherwise, Adabas returns a response code 52.

The following example shows how an insignificant null would be handled in a two-byte Adabas alphanumeric field AA when defined with and without the NN option:

Example:

An insignificant null handled in a two-byte Adabas alphanumeric field AA when defined with and without the NN option is as following:

Option	Field Definition	Null Indicator Value	Adabas Internal Representation
With NN	01,AA,2,A,NC,NN	FFFF (insignificant null)	none; response code 52 occurs
Without NN	01,AA,2,A,NC	FFFF (insignificant null)	C1

Optional Field Definition Statements

COLDE : Collation Descriptor Definition

The collation descriptor option enables descriptor values to be sorted (collated) based on a user-supplied algorithm.

The values are based on algorithms coded in special collation descriptor user exits (CDX01 through CDX08). Each collation descriptor must be assigned to a user exit, and a single user exit may handle multiple collation descriptors.

Example:

Collation descriptor Y1,File=10
Collation descriptor Y2,File=20 → Collation descriptor user exit 1 (CDX01)
Collation descriptor Y3,File=30
Collation descriptor Y5,File=9 → Collation descriptor user exit 4 (CDX04)

The exit is called whenever a collation descriptor value is to be sorted by the Adabas nucleus or by the ADACMP utility.

Input parameters supplied to the user exit are described in the *DBA Reference Manual* chapter **User Exits**. They include

- address and length of the input value
- address and length of the output buffer
- the address of the length of the returned output string

The user exit sets the length of the returned output string.

See the ADARUN parameter CDXnn in the *Adabas Operations Manual* for more information.

General Notes on Collation Descriptors

- A collation descriptor can be defined for an alphanumeric (A) or wide alphanumeric (W) parent field. The format, length, and options (except UQ and XI) are taken from the parent field defined in the COLDE parameter. The unique descriptor (UQ) and exclude index (XI) options are separately defined for the collation descriptor itself.
- A search using a collation descriptor value is performed in the same manner as for standard descriptors.

- The user is responsible for creating correct collation descriptor values. There is no standard way to check the values of a collation descriptor for completeness against the Data Storage. The maintenance utility ADAICK only checks the structure of an index, not the contents. The user must set the rules for each value definition and check the value for correctness.
- If a file contains more than one collation descriptor, the assigned exits are called in the alphabetical order of the collation descriptor names.

Collation Descriptor Syntax

A collation descriptor is defined using the following syntax:

COLDE='number**,** name**, [,UQ** [**,XI**]] **=** parent-field**'**

—where

number	is the user exit number to be assigned to the collation descriptor. The Adabas nucleus uses this number to determine the collation descriptor user exit to be called.
name	is the name to be used for the collation descriptor. The naming conventions for collation descriptors are identical to those for Adabas field names.
UQ	indicates that the unique descriptor option is to be assigned to the collation descriptor.
XI	indicates that the uniqueness of the collation descriptor is to be determined with the index (occurrence) number excluded.
parent-field	is the name of an elementary A or W field. A collation descriptor can have one parent field. The field name and address is passed to the user exit.

MU, NU, and PE options are taken from the parent field and are implicitly set in the collation descriptor.

If a parent field with the NU option is specified, no entries are made in the collation descriptor's inverted list for those records containing a null value for the field. This is true regardless of the presence or absence of values for other collation descriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated, for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Collation Descriptor Definition Example:

Field definition:

```
FNDEF='01,LN,20,A,DE,NU'     Last-Name
```

Collation descriptor definition:

```
COLDE='1,Y2=LN'
```

- Collation descriptor user exit 1 (CDX01) is assigned to this collation descriptor, and the name is Y2.
- The collation descriptor length and format are taken from the parentfield: 20 and alphanumeric, respectively. The collation descriptor is a multiple value (MU) field with null suppression (NU).
- The values for the collation descriptor are to be derived from the parentfield LN.

HYPDE : Hyperdescriptor Definition

The hyperdescriptor option enables descriptor values to be generated, based on a user-supplied algorithm.

The values are based on algorithms coded in special hyperdescriptor user exits (HEX01 through HEX31). Each hyperdescriptor must be assigned to a user exit, and a single user exit may handle multiple hyperdescriptors.

Example:

Hyperdescriptor	H1,File=20	
Hyperdescriptor	H1,File=30	→ Hyperdescriptor user exit 4 (HEX4)
Hyperdescriptor	Y7,File=30	
Hyperdescriptor	FA,File=9	→ Hyperdescriptor user exit 9 (HEX9)

The exit is called whenever a hyperdescriptor value is to be generated by the Adabas nucleus or by the ADACMP utility.

Input parameters supplied to the user exit are

- hyperdescriptor name
- file number
- addresses of fields taken from the Data Storage record, together with field name and PE index (if applicable). These addresses point to the compressed values of the fields. The names of these fields must be defined using the HYPDE parameter of ADACMP or ADAINV.

The user exit must return the descriptor value(s) (DVT) in compressed format. No value, or one or more values may be returned depending on the options (PE, MU) assigned to the hyperdescriptor.

The original ISN assigned to the input value(s) may be changed.

See the *Adabas DBA Reference Manual*, chapter **User Exits**, for more information about the hyperdescriptor user exit.

General Notes on Hyperdescriptors

- The format, the length, and the options of a hyperdescriptor are user-defined. They are not taken from the parent fields defined in the HYPDE parameter.
- A search using a hyperdescriptor value is performed in the same manner as for standard descriptors.
- The user is responsible for creating correct hyperdescriptor values. There is no standard way to check the values of a hyperdescriptor for completeness against the Data Storage. The maintenance utility ADAICK only checks the structure of an index, not the contents. The user must set the rules for each value definition and check the value for correctness.
- If a hyperdescriptor is defined as packed or unpacked format, Adabas checks the returned values for validity. The sign half-byte for packed values can contain A, C, E, F (positive) or B, D (negative). Adabas converts the sign to F or D.
- If a file contains more than one hyperdescriptor, the assigned exits are called in the alphabetical order of the hyperdescriptor names.

Hyperdescriptor Syntax

A hyperdescriptor is defined using the following syntax:

HYPDE='number**,** name**,** length**,** format [{**,**option}...] **=** {parent-field*}*,...'

—where

number — is the user exit number to be assigned to the hyperdescriptor. The Adabas nucleus uses this number to determine the hyperdescriptor user exit to be called.

name — is the name to be used for the hyperdescriptor. The naming conventions for hyperdescriptors are identical to those for Adabas field names.

length — is the default length of the hyperdescriptor.

format — is the format of the hyperdescriptor:

Format	Maximum Length
Alphanumeric (A)	253 bytes
Binary (B)	126 bytes
Fixed Point (F)	4 bytes (always 4 bytes)
Floating Point (G)	8 bytes (always 4 or 8 bytes)
Packed Decimal (P)	15 bytes
Unpacked Decimal (U)	29 bytes

Note:
Wide-character (W) format is not valid for a hyperdescriptor.

option — is an option to be assigned to the hyperdescriptor.

The following options may be used together with a hyperdescriptor:

MU multiple-value field

NU null-value suppression

PE field of a periodic group

UQ unique descriptor

parent-field — is the name of an elementary field. A hyperdescriptor can have 1–20 parent fields. The field names and addresses are passed to the user exit.

Note:
A hyperdescriptor parent-field may not have W (wide-character) format.

If a parent field with the NU option is specified, no entries are made in the hyperdescriptor's inverted list for those records containing a null value for the field. This is true regardless of the presence or absence of values for other hyperdescriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated, for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Hyperdescriptor Definition Example:

Field definitions:

```
FNDEF='01,LN,20,A,DE,NU'   Last-Name
FNDEF='01,FN,20,A,MU,NU'   First-Name
FNDEF='01,ID,4,B,NU'       Identification
FNDEF='01,AG,3,U'          Age
FNDEF='01,AD,PE'           Address
FNDEF='02,CI,20,A,NU'      City
FNDEF='02,ST,20,A,NU'      Street
FNDEF='01,FA,PE'           Relatives
FNDEF='02,NR,20,A,NU'      R-Last-Name
FNDEF='02,FR,20,A,MU,NU'   R-First-Name
```

Hyperdescriptor definition:

```
HYPDE='2,HN,60,A,MU,NU=LN,FN,FR'
```

- Hyperdescriptor user exit 2 is assigned to this hyperdescriptor, and the name is HN.
- The hyperdescriptor length is 60, the format is alphanumeric, and is a multiple-value (MU) field with null suppression (NU).
- The values for the hyperdescriptor are to be derived from fields LN, FN and FR.

The ADACMP HYPDE= statement may be continued on another line, as shown in the following example. To do so, first specify a minus (–) after a whole argument and before the closing apostrophe on the first line. Then enter the remaining positional arguments, beginning after the statement name (ADACMP) enclosed in apostrophes on the following line:

```
ADACMP HYPDE='1,HY,20,A=AA,BB,CC,-'
ADACMP                 'DD,EE,FF'
```

PHONDE : Phonetic Descriptor

The use of a phonetic descriptor in a FIND command results in the return of all the records that contain similar phonetic values. The phonetic value of a descriptor is based on the first 20 bytes of the field value. Only alphabetic values are considered; numeric values, special characters, and blanks are ignored. Lower- and uppercase alphanumeric characters are internally identical.

A phonetic descriptor is defined using the following syntax:

PHONDE='name (field)**'**

—where

name	is the name to be used for the phonetic descriptor. The naming conventions for phonetic descriptors are identical to those for Adabas field names.
field	is the name of the field to be phoneticized.

The field **must** be

- an elementary or a multiple value field; and
- defined with alphanumeric format.

The field can be a descriptor.

The field **cannot** be

- a subdescriptor, superdescriptor, or hyperdescriptor;
- contained within a periodic group;
- used as the source field for more than one phonetic descriptor.
- format W (wide-character)

If the field is defined with the NU option, no entries are made in the phonetic descriptor's inverted list for those records that contain a null value (within the byte positions specified) for the field. The format is the same as for the field.

If the field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Phonetic Descriptor Definition Example:

Field definition:

```
FNDEF='01,AA,20,A,DE,NU'
```

Phonetic definition:

```
PHONDE='PA(AA)'
```

SUBDE : Subdescriptor Definition

A subdescriptor is a descriptor created from a portion of an elementary field. The elementary field may or may not be a descriptor itself. A subdescriptor can also be used as a subfield; that is, it can be specified in the format buffer to control the record's output format.

A subdescriptor definition is entered using the following syntax:

SUBDE='name **[,UQ** [**,XI**] **]=**parent-field (begin**,** end)**'**

—where

name	is the subdescriptor name. The naming conventions for a subdescriptor are identical to those for Adabas field names.
UQ	indicates that the subdescriptor is to be defined as unique (see the definition of option UQ on page 77).
XI	indicates that the uniqueness of the subdescriptor is to be determined with the index (occurrence) number excluded.
parent-field	is the name of the field from which the subdescriptor is to be derived.
begin*	is the relative byte position within the parent field where the subdescriptor definition is to begin.
end*	is the relative byte position within the parent field where the subdescriptor definition is to end.

***** *Counting is from left to right beginning with 1 for alphanumeric or wide-character fields, and from right to left beginning with 1 for numeric or binary fields. If the parent field is defined with P format, the sign of the resulting subdescriptor value is taken from the 4 low-order bits of the low-order byte (that is, byte 1).*

A parent field of a subdescriptor can be

- a descriptor
- an elementary field
- a multiple-value field (but **not** a particular occurrence of a multiple-value field)
- contained within a periodic group (but **not** a particular occurrence of a periodic group)

A parent field or a subdescriptor **cannot** be

- a sub/super field, subdescriptor, superdescriptor, or phonetic descriptor
- format G (floating point)

If the parent field is defined with the NU option, no entries are made in the subdescriptor's inverted list for those records that contain a null value (within the byte positions specified) for the field. The format is the same as for the parent field.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

Subdescriptor Definition Example 1:

Parent-field definition:

```
FNDEF='01,AR,10,A,NU'
```

Subdescriptor definition:

```
SUBDE='SB=AR(1,5)'
```

The values for subdescriptor SB are derived from the first five bytes (counting from left to right) of all the values for the parent field AR. All values are shown in character format.

AR Values	SB Values
DAVENPORT	DAVEN
FORD	FORD
WILSON	WILSO

Subdescriptor Definition Example 2:

Parent-field definition:

```
FNDEF='02,PF,6,P'
```

Subdescriptor definition:

```
SUBDE='PS=PF(4,6)'
```

The values for subdescriptor PS are derived from bytes 4 to 6 (counting from right to left) of all the values for the parent field PF. All values are shown in hexadecimal.

PF Values	PS Values
00243182655F	02431F
00000000186F	0F (see note)
78426281448D	0784262D

Note:
If the NU option had been specified for parent field PF, no value would have been created for PS for this value.

Subdescriptor Definition Example 3:

Source-field definition:

```
FNDEF='02,PF,6,P'
```

Subdescriptor definition:

```
SUBDE='PT=PF(1,3)'
```

The values for PT are derived from bytes 1 to 3 (counting from right to left) of all the values for PF. All values are shown in hexadecimal.

PF Values	PT Values
00243182655F	82655F
00000000186F	186F
78426281448D	81448D

SUBFN : Subfield Definition

A subfield

- is a portion of an elementary field that can be read using an Adabas read command;
- cannot be updated;
- can be changed to a subdescriptor using ADAINV INVERT SUBDE=... .

A subfield definition is entered using the following syntax:

SUBFN='name = parent-field (begin**,** end)**'**

—where

name	is the subfield name. The naming conventions for a subfield are identical to those for Adabas field names.
parent-field	is the name of the field from which the subfield is to be derived.
begin*	is the relative byte position within the parent field where the subfield definition is to begin.
end*	is the relative byte position within the parent field where the subfield definition is to end.

***** *Counting is from left to right beginning with 1 for alphanumeric or wide-character fields, and from right to left beginning with 1 for numeric or binary fields. If the parent field is defined with "P" format, the sign of the resulting subfield value is taken from the 4 low-order bits of the low-order byte (that is, byte 1).*

The parent field for a subfield can be

- a multiple-value field
- within a periodic group

The parent field for a subfield **cannot** have format "G" (floating point).

Subfield Definition Example:

```
SUBFN='X1=AA(1,2)'
```

SUPDE : Superdescriptor Definition

A superdescriptor is a descriptor created from several fields, portions of fields, or a combination thereof.

Each source field (or portion of a field) used to define a superdescriptor is called a **parent**. From 2 to 20 parent fields or field portions may be used to define a superdiscriptor.

A superdescriptor may be defined as a unique descriptor.

A superdescriptor can be used as a superfield; that is, it can be specified in the format buffer to determine the record's output format.

A superdescriptor description has the following syntax:

SUPDE='name [**,UQ** [**,XI**]] = {parent-field (begin**,** end)} **,...'**

—where

name	is the superdescriptor name. The naming conventions for superdescriptors are identical to those for Adabas names.
UQ	indicates that the superdescriptor is to be defined as unique (see the definition option UQ on page 77).
XI	indicates that the uniqueness of the superdescriptor is to be determined with the index (occurrence) number excluded.
parent-field	is the name of a parent field from which a superdescriptor element is to be derived; up to 20 parent fields can be specified.
begin*	is the relative byte position within the field where the superdescriptor element begins.
end*	is the relative byte position within the field where the superdescriptor element is to end.

***** *Counting is from left to right beginning with 1 for fields defined with alphanumeric or wide-character format, and from right to left beginning with 1 for fields defined with numeric or binary format. For any parent field except those defined as "FI", any begin and end values within the range permitted for the parent field's data type are valid.*

A parent field of a superdescriptor can be

- an elementary field; or
- a maximum of one multiple-value field (but not a specific multiple-value field value);
- within a periodic group (but not a specific occurrence);
- a descriptor.

A parent field of a superdescriptor **cannot** be

- a super-, sub-, or phonetic descriptor;
- format G (floating point);
- an NC option field if another parent field is an NU option field;
- a long alphanumeric (LA) field.

If a parent field with the NU option is specified, no entries are made in the superdescriptor's inverted list for those records containing a null value for the field. This is true regardless of the presence or absence of values for other superdescriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

The total length of any superdescriptor value may not exceed 253 bytes (alphanumeric) or 126 bytes (binary).

The superdescriptor format is B (binary) if no element of the superdescriptor is derived from an A (alphanumeric) or W (wide-character) parent field; if any element of the superdescriptor is derived from an A or W parent field, the format of the superdescriptor reflects the last occurring A or W element; for example, if the last occurring A or W element is W, the format of the superdescriptor is W.

All binary format superdescriptor values are treated as unsigned numbers.

The ADACMP SUPDE= statement may be continued on another line by specifying a minus (–) after an argument just before the closing apostrophe on the first line. Then enter the remaining positional arguments enclosed in apostrophes on the following line beginning after the statement name (ADACMP). For example:

```
ADACMP SUPDE='SI=AA(10,20),BB(20,21),-'
ADACMP       'CC(12,13),DD(14,15)'
```

Superdescriptor Definition Example 1:

Field definitions:

```
FNDEF='01,LN,20,A,DE,NU'    Last-Name
FNDEF='01,FN,20,A,MU,NU'    First-Name
FNDEF='01,ID,4,B,NU'        Identification
FNDEF='01,AG,3,U'           Age
FNDEF='01,AD,PE'            Address
FNDEF='02,CI,20,A,NU'       City
FNDEF='02,ST,20,A,NU'       Street
FNDEF='01,FA,PE'            Relatives
FNDEF='02,NR,20,A,NU'       R-Last-Name
FNDEF='02,FR,20,A,MU,NU'    R-First-Name
```

Superdescriptor definition:

```
SUPDE='SD=LN(1,4),ID(3,4),AG(2,3)'
```

Superdescriptor SD is to be created. The values for the superdescriptor are to be derived from bytes 1 to 4 of field LN (counting from left to right), bytes 3 to 4 of field ID (counting from right to left), and bytes 2 to 3 of field AG (counting from right to left). All values are shown in hexadecimal.

LN	ID	AG	SD
C6D3C5D4C9D5C7	00862143	F0F4F3	C6D3C5D40086F0F4
D4D6D9D9C9E2	02461866	F0F3F8	D4D6D9D90246F0F3
D7C1D9D2C5D9	00000000	F0F3F6	No value is stored (because of ID)
404040404040	00432144	F0F0F0	No value is stored (because of LN)
C1C1C1C1C1C1	00000144	F1F1F1	C1C1C1C10000F1F1
C1C1C1C1C1C1	00860000	F0F0F0	C1C1C1C10086F0F0

The format for SD is alphanumeric since at least one element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 2:

Field definitions:

```
FNDEF='01,LN,20,A,DE,NU'    Last-Name
FNDEF='01,FN,20,A,MU,NU'    First-Name
FNDEF='01,ID,4,B,NU'        Identification
FNDEF='01,AG,3,U'           Age
FNDEF='01,AD,PE'            Address
FNDEF='02,CI,20,A,NU'       City
FNDEF='02,ST,20,A,NU'       Street
FNDEF='01,FA,PE'            Relatives
FNDEF='02,NR,20,A,NU'       R-Last-Name
FNDEF='02,FR,20,A,MU,NU'    R-First-Name
```

Superdescriptor definition:

```
SUPDE='SY=LN(1,4),FN(1,1)'
```

Superdescriptor SY is to be created from fields LN and FN (which is a multiple-value field). All values are shown in character format.

LN	FN	SY
FLEMING	DAVID	FLEMD
MORRIS	RONALD	MORRR
	RON	MORRR
WILSON	JOHN	WILSJ
	SONNY	WILSS

The format of SY is alphanumeric since at least one element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 3:

Feld definitions:

```
FNDEF='01,PN,6,U,NU'
FNDEF='01,NA,20,A,DE,NU'
FNDEF='01,DP,1,B,FI '
```

Superdescriptor definition:

```
SUPDE='SZ=PN(3,6),DP(1,1)'
```

Superdescriptor SZ is to be created. The values for the superdescriptor are to be derived from bytes 3 to 6 of field PN (counting from right to left), and byte 1 of field DP. All values are shown in hexadecimal.

PN	DP	SZ
F0F2F4F6F7F2	04	F0F2F4F604
F8F4F0F3F9F8	00	F8F4F0F300
F0F0F0F0F1F1	06	F0F0F0F006
F0F0F0F0F0F1	00	F0F0F0F000
F0F0F0F0F0F0	00	no value is stored (because of PN)
F0F0F0F0F0F0	01	no value is stored (because of PN)

The format of SZ is binary since no element is derived from a parent field defined with alphanumeric format. A null value is not stored for the last two values shown because the superdescriptor option is NU (from the PN field) and the PN field value contains unpacked zeros (X'F0'), the null value.

Superdescriptor Definition Example 4:

Field definitions:

```
FNDEF='01,PF,4,P,NU'
FNDEF='01,PN,2,P,NU'
```

Superdescriptor definition:

```
SUPDE='SP=PF(3,4),PN(1,2)'
```

Superdescriptor SP is to be created. The values for the superdescriptor are to be derived from bytes 3 to 4 of field PF (counting from right to left), and bytes 1 to 2 of field PN (counting from right to left). All values are shown in hexadecimal.

PF	PN	SP
0002463F	003F	0002003F
0000045F	043F	0000043F
0032464F	000F	No value is stored (because of PN)
0038000F	044F	0038044F

The format of SP is binary since no element is derived from a parent field defined with alphanumeric format.

Superdescriptor Definition Example 5:

Field definitions:

```
FNDEF='01,AD,PE'
FNDEF='02,CI,4,A,NU'
FNDEF='02,ST,5,A,NU'
```

Superdescriptor definition:

```
SUPDE='XY=CI(1,4),ST(1,5)'
```

Superdescriptor XY is to be created from fields CI and ST. All values are shown in character format.

CI	ST	XY
(1st occ.) BALT	(1st occ.) MAIN	BALTMAIN
(2nd occ.) CHI	(2nd occ.) SPRUCE	CHI SPRUC
(3rd occ.) WASH	(3rd occ.) 11TH	WASH11TH
(4th occ.) DENV	(4th occ.) bbbbb	No value stored (because of ST)

The format of XY is alphanumeric since at least 1 element is derived from a parent field which is defined with alphanumeric format.

SUPFN : Superfield Definition

A superfield is a field composed of several fields, portions of fields, or combinations thereof, which may be read using an Adabas read command. A superfield **cannot**

- be updated;
- comprise fields defined with the NC option if another parent field has the NU option;
- be used as a descriptor.

A superfield **can** be changed to a superdescriptor using the ADAINV utility function INVERT SUPDE=... .

A superfield is defined using the following syntax:

SUPFN='name = {parent-field (begin**,** end)}**,...'**

—where

name	superfield name. The naming conventions for superfields are identical to those for Adabas names.
parent-field	name of the field from which a superfield element is to be derived.
begin*	relative byte position within the field where the superfield element is to begin.
end*	relative byte position within the field where the superfield element is to end.

***** *Counting is from left to right beginning with 1 for fields defined with alphanumeric or wide-character format, and from right to left beginning with 1 for fields defined with numeric or binary format.*

A parent field of a superfield can be

- a multiple-value field
- contained within a periodic group

A parent field of a superfield **cannot** be format G (floating point).

The total length of any superfield value may not exceed 253 bytes (alphanumeric) or 126 bytes (binary).

The superfield format is B (binary) if no element of the superfield is derived from an A (alphanumeric) or W (wide-character) parent field; if any element of the superfield is derived from an A or W parent field, the format of the superfield reflects the last occurring A or W element; for example, if the last occurring A or W element is W, the format of the superfield is W.

Superfield Definition Example:

```
SUPFN='X2=AA(1,2),AB(1,4),AC(1,1)'
```

ADACMP COMPRESS Examples

Example 1:

```
ADACMP   COMPRESS
ADACMP   FNDEF='01,AA,7,A,DE,FI'          Field AA
ADACMP   FNDEF='01,AB,15,A,DE,MU,NU'      Field AB
ADACMP   FNDEF='01,GA'                    Group GA
ADACMP   FNDEF='02,AC,15,A,NU'            Field AC
ADACMP   FNDEF='02,AD,2,P,FI'             Field AD
ADACMP   FNDEF='02,AE,5,P,NU'             Field AE
ADACMP   FNDEF='02,AF,6,W'                Field AF
ADACMP   COLDE='7,Y1=AF'                  Collation descriptor Y1
ADACMP   SUBDE='BB=AA(1,4)'               Subdescriptor BB
ADACMP   SUPDE='CC=AA(1,4),AD(1,1)'       Superdescriptor CC
ADACMP   HYPDE='1,DD,4,A,MU=AB,AC,AD'     Hyperdescriptor DD
ADACMP   PHONDE='EE(AA)'                  Phonetic descriptor EE
ADACMP   SUBFN='FF=AA(1,2)'               Subfield FF
ADACMP   SUPFN='GG=AA(1,4),AD(1,1)'       Superfield GG
```

- Field AA is defined as level 1, 7 bytes alphanumeric, descriptor, fixed storage option.
- Field AB is defined as level 1, 15 bytes alphanumeric, descriptor, multiple value field, null value suppression.
- GA is a group containing fields AC, AD and AE.
- BB is a subdescriptor (positions 1–4 of field AA).
- CC is a superdescriptor (positions 1–4 of field AA and position 1 of field AD).
- DD is a hyperdescriptor consisting of fields AB, AC and AD. DD is assigned hyperexit 1.
- EE is a phonetic descriptor derived from field AA.
- FF is a subfield (positions 1–2 of field AA).
- GG is a superfield (positions 1–4 of AA and position 1 of AD).
- Y1 is a collation descriptor for AF and is assigned to collation descriptor user exit 7 (CDX07).

Example 2:

```
ADACMP   COMPRESS
ADACMP   FORMAT='AG,6,U,AF,4X,AA,'    input record format
ADACMP   FORMAT='AB,AC'               continuation of FORMAT statement
ADACMP   FNDEF='01,AA,10,A,NU'        field definitions
ADACMP   FNDEF='01,AB,7,U,NU'
ADACMP   FNDEF='01,AF,5,P,NU'
ADACMP   FNDEF='01,AG,12,P,NU,DE'
ADACMP   FNDEF='01,AC,3,A,NU,DE'
```

The input record format is provided explicitly using the FORMAT parameter. ADACMP uses this format as the basis for processing fields from the input record. The FDT for the file corresponds to the structure specified in the FNDEF statements.

Example 3:

```
ADACMP COMPRESS
ADACMP FORMAT='AG,AF,4X,AA,AB,AC'     input record format
ADACMP FDT=8                          FDT same as file 8
```

The input record format is provided explicitly using the FORMAT parameter. The FDT to be used is the same as that currently defined for Adabas file 8.

Example 4:

```
ADACMP   COMPRESS NUMREC=2000,USERISN
ADACMP   FNDEF='01,AA,7,A,DE,FI'            Field AA
ADACMP   FNDEF='01,AB,15,A,DE,MU,NU'        Field AB
```

The number of input records to be processed is limited to 2,000. The ISN for each record is to be provided by the user.

Example 5:

```
ADACMP   COMPRESS RECFM=FB,LRECL=100
ADACMP   FNDEF='01,AA,7,A,DE,FI'            Field AA
ADACMP   FNDEF='01,AB,15,A,DE,MU,NU'        Field AB
```

A VSE input file contains fixed length (blocked) records. The record length is 100 bytes.

DECOMPRESS : Decompress File(s)

The DECOMPRESS function decompresses data either

- from output unloaded by the ADAULD UNLOAD utility function; or
- directly from a single compressed Adabas file when the file number is specified with the INFILE parameter.

When decompressing data directly from the INFILE file, DECOMPRESS first performs an ADAULD UNLOAD/MODE=SHORT function. This can save time over separate ADAULD and ADACMP DECOMPRESS operations.

ADACMP DECOMPRESS [**CODE**=cipher-code]
[**FORMAT**=output-record-format-definition]
[**INFILE**=file-number]
[**ETID**=owner-id]
[**LPB**={prefetch-buffer-size | based-on-ADARUN-lu}]
[**PASSWORD**='password']
[**SORTSEQ**={descriptor | **ISN** | physical-sequence}]
[**UTYPE**={**EXF** | **EXU**}]
[**ISN**]
[**NOUSERABEND**]
[**NUMREC**={number-of-records | all-records}]
[**TRUNCATE**]
[**UACODE**=user-alpha-key]
[**UWCODE**=user-wide-key]
[**UARC**=[architecture-key | **2**}]

Optional Parameters and Subparameters

CODE : Cipher Code

If the file to be decompressed is ciphered, the cipher code that was used when the file was compressed must be specified with this parameter. See the *Adabas Security Manual* for additional information on the use of ciphering.

ETID : Multiclient File Owner ID

ETID specifies an owner ID for a multiclient file specified by INFILE. ADACMP DECOMPRESS selectively decompresses only those records in the multiclient file assigned to the owner ID specified by ETID. The ETID value must be the same as that assigned to the records when they were loaded into the multiclient file.

FORMAT : Output Record Format Definition

FORMAT allows decompression to a format other than that specified by the FDT. It can be used to change the FDT of an existing file and, in particular, the structure of a periodic (PE) group.

The FORMAT parameter syntax is the same as the format buffer syntax used for read commands except that text cannot be inserted (text is not compressible/decompressible); see the *Adabas Command Reference Manual* for more information.

Note:
The FORMAT parameter does not check whether all related data fields have been processed during decompression. For example, if a multiple-value (MU) field defined as

01,AA,8,A,MU

—has five occurrences, and the ADACMP DECOMPRESS FORMAT parameter specifies

AA1-4

—then only the first four AA field values are decompressed; no indication is given regarding the fifth field value. This also applies to PE field occurrences and length overrides.

INFILE : Number of File to Be Decompressed

The INFILE parameter allows you to decompress a file without first unloading it with the ADAULD utility. If the INFILE parameter is not specified, the input is read from a sequential (DD/EBAND) file. With the ETID parameter, INFILE permits selectively decompressing records from a multiclient file. When decompressing multiclient files, refer to the section **Decompressing Multiclient Files** on page 107.

ISN : Include ISN in Decompressed Output

The ISN of each record is to be included with each decompressed record output. If this parameter is omitted, the ISN will not be included with each record.

LPB : Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer for the ADACMP DECOMPRESS INFILE function. The maximum value is 32,760 bytes. The default is calculated by Adabas, depending on the ADARUN LU value in effect for the nucleus.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMREC : Number of Records to Be Processed

NUMREC specifies the number of input records to be processed. If this parameter is omitted, all input records contained on the input dataset are processed.

Use of NUMREC is recommended for the initial ADACMP execution if a large number of records are contained on the input dataset. This avoids unneeded processing of all records when a field definition error or invalid input data causes a large number of rejected records. NUMREC is also useful for creating small files for test purposes.

PASSWORD : Password for INFILE

The PASSWORD parameter must specify the correct password if the file is to be decompressed directly from a password-protected Adabas file.

SORTSEQ : Processing Sequence for INFILE File

SORTSEQ determines the sequence in which the file is processed. If this parameter is omitted, the records are processed in physical sequence. SORTSEQ can be specified only when INFILE is also specified.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. **Do not** use a null-suppressed descriptor field, a hyperdescriptor, a phonetic descriptor, a multiple-value descriptor field, or a descriptor contained in a periodic group.

Note:
Even when the descriptor field is not null-suppressed, the record is ***not*** *represented in the inverted list if the descriptor field or a field following it has never been initialized (held a value). Therefore, the record will be dropped when the utility is executed.*

If ISN is specified, the file is processed in ascending ISN sequence. For the Adabas checkpoint or security file, only SORTSEQ=ISN is allowed.

TRUNCATE : Truncate Excess Alphanumeric Characters

The TRUNCATE parameter enables truncation of compressed alphanumeric data during decompression. When TRUNCATE is specified and ADACMP DECOMPRESS operation finds an alphanumeric field containing more characters than the FDT description allows for the field, the extra characters are truncated. If TRUNCATE is not specified, alphanumeric records with extra characters are written to the DD/FEHL dataset. Non-alphanumeric fields cannot be truncated.

UACODE : Encoding Protocol for Output Alphanumeric Fields

UACODE defines the encoding of the sequential output of alphanumeric fields. This parameter allows you to override the user encoding for alphanumeric fields passed in the header of the compressed sequential input.

UARC : Architecture for Output Uncompressed User Data

The UARC parameter specifies the architecture of the sequential output of the uncompressed user data. This parameter allows you to override the user encoding passed in the header of the compressed sequential input.

The ‘userdata-architecture-key’ is an integer which is the sum of the following numbers:

byte order	b=0	high-order byte first
	b=1	low-order byte first
encoding family	e=0	ASCII encoding family
	e=2	EBCDIC encoding family (default)
floating-point format	f=0	IBM370 floating-point format
	f=4	VAX floating-point format
	f=8	IEEE floating-point format

The default is ARC = b + e + f = 2; that is, high-order byte first; EBCDIC encoding family; and IBM370 floating-point format (b=0; e=2; f=0).

User data from an Intel386 PC provides the example: b=1; e=0; f=8; or ARC=9.

UTYPE : User Type

The user type to be in effect when unloading the file specified by INFILE. Allowed values are

EXF no access/update allowed for other users of the file.

EXU access only is allowed for other users of the file. EXU is the default.

UWCODE : Encoding Protocol for Output Wide-Character Fields

UWCODE defines the encoding of the sequential output of wide-character fields. This parameter allows you to override the user encoding for wide-character fields passed in the header of the compressed sequential input.

Decompressing Multiclient Files

ADACMP decompresses Adabas data to a sequential user file. The DECOMPRESS function can decompress records selectively if the INFILE parameter specifies a multiclient file and a valid ETID value is specified.

The DECOMPRESS function skips the owner ID, if present. The output of a DECOMPRESS operation on a multiclient file contains neither owner ID nor any ETID information.

If the INFILE parameter specifies a multiclient file for the DECOMPRESS function, you can use the ETID parameter to limit decompression to records for a specific user only. ADACMP then reads and decompresses records only for the specified user. If the ETID parameter is not specified when decompressing a multiclient file, all records in the file are decompressed.

Example:

Only records owned by USER1 from file 20 are decompressed to a sequential output file:

```
ADACMP  DECOMPRESS  INFILE=20,ETID=USER1
```

ADACMP DECOMPRESS Examples

Example 1:

The DECOMPRESS function is to be executed. The input dataset to be used is the output of a previous execution of the ADAULD utility:

```
ADACMP DECOMPRESS
```

Example 2:

Adabas file 23 is to be decompressed. The ISN of each record is to be included in the decompressed output:

```
ADACMP DECOMPRESS INFILE=23,ISN
```

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADACMP with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems and shows examples of each of the job streams.

Note:
When the recovery log is active, sequential datasets used by the utilities whose runs are logged on the RLOG must be kept and made available for any recovery operation; for example, the DD/EBAND input to an ADALOD LOAD operation.

User Exits with ADACMP

Compression with User Exit

User exit 6 can be used to perform user processing on a record before it is processed by the ADACMP COMPRESS utility. See the *DBA Reference Manual* for more information.

If user exit 6 is to be used during ADACMP execution, the specified user exit routine must be loadable at execution time; that is, it must be assembled and linked into the Adabas

- load library (or any library concatenated with it) for BS2000, OS/390, VM/ESA.
- core image library or any library contained in the core image library search chain for VSE/ESA.

The ADACMP COMPRESS utility job must specify

ADARUN UEX6=exit-name

—where

exit-name is the name of a user routine that gets control at the user exit; the name can be up to 8 characters long.

Collation with User Exit

If a collation user exit is to be used during ADACMP execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding support (UES), the format of the collation descriptor user exit parameter is

ADARUN CDXnn**=**exit-name

—where

nn is the number of the collation descriptor exit, a two-digit decimal integer in the range 01–08 inclusive.

exit-name is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *DBA Reference Manual* for more information.

BS2000

Dataset	Link Name	Storage	More Information
User input data (COMPRESS function)	DDEBAND	tape/disk	
Compressed data (DECOMPRESS function)	DDEBAND	tape/disk	Not used if the parameter INFILE is used
Compressed data (COMPRESS function)	DDAUSBA	tape/disk	
Decompressed data (DECOMPRESS function)	DDAUSBA	tape/disk	
Rejected data	DDFEHL	tape/disk	
ECS encoding objects	DDECSOJ	tape/disk	Required for universal encoding support (UES)
ADARUN parameters	SYSDTA/DDCARD		*Operations Manual*
ADACMP parameters and data definitions	SYSDTA/DDKARTE		*Utilities Manual*
ADARUN messages	SYSOUT/ DDPRINT	printer/disk	*Messages and Codes*
ADACMP report	SYSLST/ DDDRUCK	printer/disk	*Messages and Codes*

JCL Examples (BS2000)

ADACMP COMPRESS

In SDF Format:

```
/.ADACMP LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A C M P COMPRESS
/REMARK *
/DELETE-FILE CMP.AUS
/SET-JOB-STEP
/DELETE-FILE CMP.FEHL
/SET-JOB-STEP
/CREATE-FILE CMP.AUS,PUB(SPACE=(48,48)
/SET-JOB-STEP
/CREATE-FILE CMP.FEHL,PUB(SPACE=(48,48))
/SET-JOB-STEP
```

```
/ASS-SYSLST L.CMP
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDEBAND,CMP.EIN
/SET-FILE-LINK DDAUSBA,CMP.AUS
/SET-FILE-LINK DDFEHL,CMP.FEHL
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADACMP,DB=yyyyy,IDTNAME=ADABAS5B
ADACMP COMPRESS NUMREC=1000,FDT=1,USERISN,DEVICE=dddd,eeee
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADACMP LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A C M P COMPRESS
/REMARK *
/ER CMP.AUS
/STEP
/ER CMP.FEHL
/STEP
/SYSFILE SYSLST=L.CMP
/FILE ADA.MOD,LINK=DDLIB
/FILE CMP.EIN,LINK=DDEBAND
/FILE CMP.AUS,LINK=DDAUSBA,SPACE=(48,48)
/FILE CMP.FEHL,LINK=DDFEHL,SPACE=(48,48)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADACMP,DB=yyyyy,IDTNAME=ADABAS5B
ADACMP COMPRESS NUMREC=1000,FDT=1,USERISN,DEVICE=dddd,eeee
/LOGOFF NOSPOOL
```

ADACMP DECOMPRESS

In SDF Format:

```
/.ADACMP LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A C M P DECOMPRESS
/REMARK *
```

```
/DELETE-FILE CMP.AUS
/SET-JOB-STEP
/DELETE-FILE CMP.FEHL
/SET-JOB-STEP
/CREATE-FILE CMP.AUS,PUB(SPACE=(48,48))
/SET-JOB-STEP
/CREATE-FILE CMP.FEHL,PUB(SPACE=(48,48))
/SET-JOB-STEP
/ASS-SYSLST L.DEC
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDEBAND,CMP.EIN
/SET-FILE-LINK DDAUSBA,CMP.AUS
/SET-FILE-LINK DDFEHL,CMP.FEHL
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADACMP,DB=yyyyy,IDTNAME=ADABAS5B
ADACMP DECOMPRESS
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADACMP LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A C M P DECOMPRESS
/REMARK *
/ER CMP.AUS
/STEP
/ER CMP.FEHL
/STEP
/SYSFILE SYSLST=L.CMP.DEC


/FILE ADA.MOD,LINK=DDLIB
/FILE CMP.EIN,LINK=DDEBAND
/FILE CMP.AUS,LINK=DDAUSBA,SPACE=(48,48)
/FILE CMP.FEHL,LINK=DDFEHL,SPACE=(48,48)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADACMP,DB=yyyyy,IDTNAME=ADABAS5B
ADACMP DECOMPRESS
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
User input data (COMPRESS function)	DDEBAND	tape/disk	
Compressed data (DECOMPRESS function)	DDEBAND	tape/disk	Not used if the parameter INFILE is specified
Compressed data (COMPRESS function)	DDAUSBA	tape/disk	
Decompressed data (DECOMPRESS function)	DDAUSBA	tape/disk	
Rejected data	DDFEHL	tape/disk	
ECS encoding objects	DDECSOJ	tape/disk	Required for universal encoding support (UES)
ADACMP report	DDDRUCK	printer	
ADARUN messages	DDPRINT	printer	
ADARUN parameters	DDCARD	reader	
ADACMP parameters and data definitions	DDKARTE	reader	

JCL Examples (OS/390 or z/OS)

In the MVSJOBS dataset, refer to ADACMP for the COMPRESS example and ADACMPD for the DECOMPRESS example.

ADACMP COMPRESS

```
//ADACMP    JOB
//*
//*   ADACMP COMPRESS
//*      COMPRESS A FILE
//*
//CMP       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
```

```
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.INPUT,UNIT=TAPE, <===
//          VOL=SER=TAPE01                                      <===
//DDAUSBA   DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.COMP01,UNIT=DISK, <==
//          VOL=SER=DISK01,SPACE=(TRK,(200,10),RLSE)
//DDFEHL    DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.FEHL,UNIT=DISK, <===
//          VOL=SER=DISK01,SPACE=(TRK,1)
//DDCARD    DD   *
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADACMP COMPRESS FILE=1
ADACMP FNDEF='01,AA,008,B,DE'
ADACMP FNDEF='01,BA,020,A,NU,DE'
ADACMP FNDEF='01,BB,015,A,NU,DE'
ADACMP FNDEF='01,BC,001,A,FI'
ADACMP FNDEF='01,CA,001,A,NU,DE'
ADACMP FNDEF='01,CB,002,U,NU,DE'
ADACMP FNDEF='01,CC,010,A,NU,DE'
ADACMP FNDEF='01,CD,002,U,NU,DE'
ADACMP FNDEF='01,DA,005,U,NU'
ADACMP FNDEF='01,DB,020,A,NU,DE'
ADACMP FNDEF='01,DC,015,A,NU,DE'
ADACMP FNDEF='01,DD,002,A,NU,DE'
ADACMP FNDEF='01,DE,005,U,NU,DE'
ADACMP FNDEF='01,DF,008,A,NU,DE'
ADACMP FNDEF='01,FA,020,A,NU,DE'
ADACMP FNDEF='01,FB,006,U,NU,DE'
ADACMP FNDEF='01,FC,006,U,NU'
ADACMP FNDEF='01,GA,002,U,NU'
ADACMP FNDEF='01,HA,002,U,NU'
ADACMP FNDEF='01,IA,002,U,NU'


ADACMP FNDEF='01,KA,002,U,NU'
ADACMP FNDEF='01,LA,030,A,NU,DE'
ADACMP SUBDE='SB=DE(3,5)'
ADACMP SUPDE='SP=CA(1,1),CB(1,2),CD(1,2)'
ADACMP PHONDE='PA(BA)'
/*
//
```

ADACMP DECOMPRESS

```
//ADACMP    JOB
//*
//*   ADACMP COMPRESS
//*      DECOMPRESS A FILE
//*
//DECMP     EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <===DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <===WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.COMP01,UNIT=TAPE,
//          VOL=SER=TAPE01
//DDAUSBA   DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.DECOMP01,UNIT=DISK,==
//          VOL=SER=DISK01,SPACE=(TRK,(200,10),RLSE)
//DDFEHL    DD   DISP=(NEW,KEEP),DSN=EXAMPLE.DByyyyy.FEHL,UNIT=DISK,    =
//          VOL=SER=DISK01,SPACE=(TRK,1)
//DDCARD    DD   *
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADACMP DECOMPRESS INFILE=1
00000100
/*
//
```

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
User input data (COMPRESS function)	DDEBAND	tape/disk	
Compressed data (DECOMPRESS function)	DDEBAND	tape/disk	Not used if the parameter INFILE is specified
Compressed data (COMPRESS function)	DDAUSBA	tape/disk	
Decompressed data (DECOMPRESS function)	DDAUSBA	tape/disk	
Rejected data	DDFEHL	tape/disk	
ECS encoding objects	DDECSOJ	tape/disk	Required for universal encoding support
ADACMP report	DDDRUCK	disk/terminal/printer	
ADARUN messages	DDPRINT	disk/terminal/printer	
ADARUN parameters	DDCARD	disk/terminal/reader	
ADACMP control cards and data definitions	DDKARTE	disk/terminal/reader	

JCL Example (VM/ESA or z/VM)

ADACMP COMPRESS

```
DATADEF DDEBAND,DSN=FILE015.CMPD015,MODE=A
DATADEF DDAUSBA,DSN=FILE015.LODD015,MODE=A
DATADEF DDFEHL,DSN=FILE015.CMPERROR,MODE=A
DATADEF DDDRUCK,DSN=ADACMP.DDDRUCK,MODE=A
DATADEF DDPRINT,DSN=ADACMP.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDCARD,DSN=RUNCMP.CONTROL,MODE=A
DATADEF DDKARTE,DSN=FILE001.CMPC015,MODE=A
ADARUN
```

Contents of RUNCMP CONTROL A1:

```
ADARUN PROG=ADACMP,DEVICE=dddd,DB=yyyyy
```

Contents of FILE001 CMPC015 A1:

```
ADACMP COMPRESS NUMREC=1000,FDT=1,USERISN,DEVICE=dddd,eeee
```

VSE/ESA

File	File Name	Storage	Logical Unit	More Information
User input data (COMPRESS function)	EBAND	tape disk	SYS010 *	
Compressed data (DECOMPRESS function)	EBAND	tape disk	SYS010 *	Not used if parameter INFILE is specified
Compressed data (COMPRESS function)	AUSBA	tape disk	SYS016 *	
Decompressed data (DECOMPRESS function)	AUSBA	tape disk	SYS016 *	
Rejected data	FEHL	tape disk	SYS017 *	
ECS encoding objects	ECSOJ	tape disk	SYS020 *	Required for universal encoding support
ADACMP report	–	printer	SYS009	
ADARUN messages	–	printer	SYSLST	
ADARUN parameters	– CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADACMP control cards and data definitions	–	reader	SYSIPT	

***** Any programmer logical unit may be used.

JCS Examples (VSE/ESA)

See appendix B for descriptions of the VSE procedures.

Refer to member ADACMP.X for the COMPRESS example and member ADACMPD.X for the DECOMPRESS example.

ADACMP COMPRESS

```
* $$ JOB JNM=ADACMP,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       COMPRESS A FILE
// JOB ADACMP
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL EBAND,'EXAMPLE.DByyyyy.UNCOMP01'
// MTC REW,SYS010
// DLBL AUSBA,'EXAMPLE.DByyyyy.COMP01',,SD
// EXTENT SYS016,,,,sssss,nnnnn
// ASSGN SYS016,DISK,VOL=DISK01,SHR
// DLBL FEHL,'EXAMPLE.DByyy.FEHL',,SD
// EXTENT SYS017,,,,sssss,nnnnn
// ASSGN SYS017,DISK,VOL=DISK02,SHR
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADACMP COMPRESS FILE=1
ADACMP FNDEF='01,AA,008,B,DE'
ADACMP FNDEF='01,BA,020,A,NU,DE'
ADACMP FNDEF='01,BB,015,A,NU,DE'
ADACMP FNDEF='01,BC,001,A,FI'
ADACMP FNDEF='01,CA,001,A,NU,DE'
ADACMP FNDEF='01,CB,002,U,NU,DE'
ADACMP FNDEF='01,CC,010,A,NU,DE'
ADACMP FNDEF='01,CD,002,U,NU,DE'
ADACMP FNDEF='01,DA,005,U,NU'
ADACMP FNDEF='01,DB,020,A,NU,DE'
ADACMP FNDEF='01,DC,015,A,NU,DE'
ADACMP FNDEF='01,DD,002,A,NU,DE'
ADACMP FNDEF='01,DE,005,U,NU,DE'
ADACMP FNDEF='01,DF,008,A,NU,DE'
ADACMP FNDEF='01,FA,020,A,NU,DE'
ADACMP FNDEF='01,FB,006,U,NU,DE'
ADACMP FNDEF='01,FC,006,U,NU'
```

```
ADACMP FNDEF='01,GA,002,U,NU'
ADACMP FNDEF='01,HA,002,U,NU'
ADACMP FNDEF='01,IA,002,U,NU'
ADACMP FNDEF='01,KA,002,U,NU'
ADACMP FNDEF='01,LA,030,A,NU,DE'
ADACMP SUBDE='SB=DE(3,5)'
ADACMP SUPDE='SP=CA(1,1),CB(1,2),CD(1,2)'
ADACMP PHONDE='PA(BA)'
/*
/&
* $$ EOJ
```

ADACMP DECOMPRESS

```
* $$ JOB JNM=ADACMPD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       DECOMPRESS A FILE
// JOB ADACMPD
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,TAPE
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL EBAND,'EXAMPLE.DByyyyy.COMP01'
// MTC REW,SYS010
// DLBL AUSBA,'EXAMPLE.DByyyyy.DECOMP01',,SD
// EXTENT SYS016,,,,sssss,nnnnn
// ASSGN SYS016,DISK,VOL=DISK01,SHR
// DLBL FEHL,'EXAMPLE.DByyy.FEHL',,SD
// EXTENT SYS017,,,,sssss,nnnnn
// ASSGN SYS017,DISK,VOL=DISK02,SHR
*
* ********************************************************************
*     REMEMBER TO CUSTOMIZE PARAMETERS OF ADABAS UTILITY
* ********************************************************************
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADACMP,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADACMP DECOMPRESS INFILE=1
/*
/&
* $$ EOJ
```

4

ADACNV : DATABASE CONVERSION

Functional Overview

The ADACNV utility converts (CONVERT) an Adabas database from version 5.2 or above to a higher version, and the reverse (REVERT).

Warning:
Before you convert a database, you must terminate all active nucleus or utility jobs normally.

To ensure database integrity, ADACNV writes changed blocks first to intermediate storage; that is, to the sequential dataset DD/FILEA. After all changed blocks have been written out to DD/FILEA, a “point of no return” is reached and the changed blocks are written to the database. If ADACNV terminates abnormally after the “point of no return”, the RESTART parameter can be used to begin the ADACNV run by reading the contents of DD/FILEA and writing them out to the database.

Important:
To ensure database integrity, DD/FILEA must be defined permanently and be deleted only after ADACNV has completed successfully. The DD/FILEA dataset must not be defined as a temporary dataset that is automatically deleted at the end of the job.

The TEST parameter is provided to check the feasibility of a conversion or reversion without writing any changes to the database. It is therefore not necessary to terminate all activity on the database before running ADACNV when you use the TEST parameter.

Database Status

Internally, the utility converts or reverts one version at a time until the target version is attained. It is therefore important to ensure that all requirements for conversion or reversion between the current and target database levels have been met before you execute ADACNV without the TEST parameter.

Before a conversion or reversion begins, ADACNV checks the status of the database:

- The DIB must be empty; that is, no Adabas nucleus or utility may be active or have been terminated abnormally. If RESTART is specified, the DIB must contain the entry of ADACNV, which includes a time stamp.

- For conversion from version 5.2, the checkpoint block 8 must have enough free space to accommodate the expanded 24-byte header used for version 5.3 and above. For reversion to version 5.2, the checkpoint blocks 20–24 must be empty.
- The Work dataset must not have a pending autorestart.

If this check is successful, ADACNV locks the database and creates a DIB entry.

For reversions, ADACNV checks whether any features are used that do not exist in the target version and returns a message if any are found.

Procedure

The procedure for converting or reverting an Adabas database is as follows:

1. If the nucleus is active, use ADAEND to stop it.
2. Use ADARES PLCOPY/CLCOPY to copy all protection and command logs.

   For your installation, this may be done automatically with user exit 2.

   Wait until the logs have been copied.
3. Optionally, back up the database (full or delta).
4. Execute the ADACNV utility.
5. Start the nucleus of the version to which you have converted or reverted.

CONVERT : Convert Database to Higher Version

The CONVERT function starts from the Adabas version of the last nucleus session.

ADACNV CONVERT [**IGNPPT**]
[**NOUSERABEND**]
[**PLOGDEV=**{multiple-PLOG-device-type | ADARUN-device}]
[**RESTART**]
[**TEST**]
[**TOVERS=** {target-version | ADACNV-version}]

Optional Parameters

IGNPPT : Ignore Parallel Participant Table PLOG Entries

When converting from a version of Adabas that uses the parallel participant table (PPT) structure to a higher version of Adabas, an error is printed and conversion fails if the system detects one or more protection logs (PLOGs) from the current version that have not been copied/merged.

If IGNPPT is specified, the utility will continue processing in spite of the uncopied/unmerged PLOGs.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PLOGDEV : Multiple PLOG Device Type

PLOGDEV specifies the physical device type on which the multiple protection log datasets to be converted are contained. If PLOGDEV is not specified, the device type specified by the ADARUN DEVICE parameter is used.

RESTART : Rerun after Point of No Return

If ADACNV terminates abnormally after the “point of no return”, that is, after all changed blocks have been written to DD/FILEA, the RESTART parameter instructs ADACNV to begin its run by reading the contents of DD/FILEA and continue by writing them to the database.

TEST : Test Conversion

The TEST parameter tests the feasibility of the conversion operation without actually writing any changes to the database.

TOVERS : Target Version

The version of Adabas database (version and revision level) to achieve at the end of the ADACNV run. If the TOVERS parameter is

- specified, it must be a version higher than the source version.
- not specified, ADACNV uses its own version as the target version.

The version format is **vr** indicating the version and revision level; for example, **74**.

Conversion Considerations

The following is an overview of the conversion steps performed by ADACNV.

All Versions

- The data protection area on the Work dataset and the multiple PLOG datasets (if supplied) are cleared to binary zeros.

From Version 5.2 to 5.3

- The new checkpoint file FDT is installed.
- For a security file, any search-by-value criteria are adjusted to the new internal search structure.

From Version 5.3 to 6.1

- The free space table (FST) is converted from 3- to 4-byte RABN. If an FST RABN overflow occurs, the smallest FST extent is removed. This is repeated until the FST fits into the ASSO block. An appropriate message is printed.
- Unused RABN chains are converted from 3- to 4-byte RABNs for each loaded file.
- If a block of unreadable blocks (BUB) exists, it is converted from 3- to 4-byte RABN structure.
- The new security file FDT is installed.
- Any Delta Save Facility DLOG area header is set to the correct version. If the Delta Save Facility logging status is “enabled”, it is set to “disabled” and an appropriate message is printed.

From Version 6.1 to 6.2

- Any Delta Save Facility DLOG area header is set to the correct version.

From Version 6.2 to 7.1

- Any Delta Save Facility DLOG area header is set to the correct version.

Example

```
ADACNV CONVERT TOVERS=71
```

The version of Adabas selected in the last nucleus session is to be converted to a version 7.1 database.

REVERT : Revert Database to Lower Version

The REVERT function starts from the Adabas version of the last nucleus session.

ADACNV REVERT **TOVERS=**target-version
[**IGNPPT**]
[**NOUSERABEND**]
[**PLOGDEV=**{multiple-PLOG-device-type | ADARUN-device}]
[**RESTART**]
[**TEST**]

Essential Parameter and Subparameter

TOVERS : Target Version

The version of Adabas database (version and revision level) to achieve at the end of the ADACNV run. The TOVERS parameter value must be a version lower than the source version.

The version format is **vr** indicating the version and revision level; for example, **61**.

Optional Parameter

IGNPPT : Ignore Parallel Participant Table PLOG Entries

When reverting from a version of Adabas that uses the parallel participant table (PPT) structure to a lower version of Adabas, an error is printed and conversion fails if the system detects one or more protection logs (PLOGs) from the current version that have not been copied/merged.

If IGNPPT is specified, the utility will continue processing in spite of the uncopied/unmerged PLOGs.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PLOGDEV : Multiple PLOG Device Type

PLOGDEV specifies the physical device type on which the multiple protection log datasets to be reverted is contained. If PLOGDEV is not specified, the device type specified by the ADARUN DEVICE parameter is used.

RESTART : Rerun after Point of No Return

If ADACNV terminates abnormally after the “point of no return”, that is, after all changed blocks have been written to DD/FILEA, the RESTART parameter instructs ADACNV to begin its run by reading the contents of DD/FILEA and continue by writing them to the database.

TEST : Test Conversion

The TEST parameter tests the feasibility of the reversion operation without actually writing any changes to the database.

Reversion Considerations

The following is an overview of the reversion steps performed by ADACNV.

All Versions

- Reversion is not possible if any Adabas feature is used in the current version that is not supported in the target version. This statement applies to all Adabas features that affect the structure of the database.

From Version 7.1 to 6.2

- Version 7.1 extends the free space table (FST) from one RABN (RABN 10) to five RABNs (RABNs 10–14). ADACNV checks whether all FST entries fit into one RABN. If not, the smallest FST extent is removed. This is repeated until the FST fits into one ASSO block. An appropriate message is printed.
- Any Delta Save Facility DLOG area header is set to the correct version.

From Version 6.2 to 6.1

- Any Delta Save Facility DLOG area header is set to the correct version.

From Version 6.1 to 5.3

- The free space table (FST) is reverted from 4- to 3-byte RABNs.
- Unused RABN chains are reverted from 4- to 3-byte RABNs for each loaded file.
- Any Delta Save Facility DLOG area header is set to the correct version. If the Delta Save Facility logging status is “enabled”, it is set to “disabled” and an appropriate message is printed.
- If a block of unreadable blocks (BUB) exists, it is reverted from 4- to 3-byte RABN structure.
- The older security file FDT is installed.

From Version 5.3 to 5.2

- The older checkpoint file FDT is installed.
- Any security-by-value criteria **will not revert**. This means that a security file with security-by-value criteria must be deleted before the reversion and defined again with version 5.2.

Example

ADACNV REVERT TOVERS=53

The Adabas version of the last run of the nucleus is to be converted back (reverted) to a version 5.3 Adabas database.

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADACNV with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems, and shows examples of each of the job streams.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1	disk	
Multiple protection logs	DDPLOGRn	disk	
Intermediate storage	DDFILEA	tape/disk	see note
ADARUN parameters	SYSDTA/ DDCARD		*Operations Manual*
ADACNV parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADACNV messages	SYSLST/ DDDRUCK		*Messages and Codes*

Note:
The intermediate storage is read an undefined number of times. If this storage is on tape/cassette, it is necessary to use the ADARUN parameter TAPEREL=NO to prevent the tape from being released. Software AG then recommends that you put a tape release command in the job to free the tape/cassette unit when the job has finished. See the example following.

ADACNV JCL Example (BS2000)

With Intermediate Disk File Storage

In SDF Format:

```
/.ADACNV LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A C N V CONVERT THE DATABASE TO NEW VERSION
/REMARK *
/DELETE-FILE ADAyyyyy.FILEA
/SET-JOB-STEP
/CREATE-FILE ADAyyyyy.FILEA,PUB(SPACE=(4800,480))
/SET-JOB-STEP
/ASS-SYSLST L.CNV.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
/SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOGR1,SHARE-UPD=YES
/SET-FILE-LINK DDPLOGR2,ADAyyyyy.PLOGR2,SHARE-UPD=YES
/SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADACNV,DB=yyyyy,IDTNAME=ADABAS5B
ADACNV CONVERT TOVERS=vr
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADACNV LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A C N V CONVERT THE DATABASE TO NEW VERSION
/REMARK *
/SYSFILE SYSLST=L.CNV.DATA
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA,LINK=DDDATAR1,SHARUPD=YES
/FILE ADAyyyyy.WORK,LINK=DDWORKR1,SHARUPD=YES
/FILE ADAyyyyy.PLOGR1,LINK=DDPLOGR1,SHARUPD=YES
/FILE ADAyyyyy.PLOGR2,LINK=DDPLOGR2,SHARUPD=YES
/FILE ADAyyyyy.FILEA,LINK=DDFILEA,SPACE=(4800,480)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADACNV,DB=yyyyy,IDTNAME=ADABAS5B
ADACNV CONVERT TOVERS=vr
/LOGOFF NOSPOOL
```

With Intermediate Tape/Cassette File Storage

In SDF Format:

```
/.ADACNV LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A C N V CONVERT THE DATABASE TO NEW VERSION
/REMARK * INTERMEDIATE TAPE/CASSETTE STORAGE
/REMARK *
/DELETE-FILE ADAyyyyy.FILEA
/SET-JOB-STEP
/CREATE-FILE ADAyyyyy.FILEA,TAPE(DEV-TYPE=T-C1,VOL=ADA001)
/SET-JOB-STEP
/ASS-SYSLST L.CNV.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
/SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOGR1,SHARE-UPD=YES
/SET-FILE-LINK DDPLOGR2,ADAyyyyy.PLOGR2,SHARE-UPD=YES
/SET-FILE-LINK DDFILEA,ADAyyyyy.FILEA,TAPE(FILE-SEQ=1),OPEN-MODE=OUTIN
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADACNV,DB=yyyyy,IDTNAME=ADABAS5B,TAPEREL=NO
ADACNV CONVERT TOVERS=vr
/SET-JOB-STEP
/REMARK * NOW RELEASE THE TAPE
/REM-FILE-LINK DDFILEA,UNL-REL-TAPE=YES
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADACNV LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A C N V CONVERT THE DATABASE TO NEW VERSION
/REMARK * INTERMEDIATE TAPE/CASSETTE STORAGE
/REMARK *
/SYSFILE SYSLST=L.CNV.DATA
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA,LINK=DDDATAR1,SHARUPD=YES
/FILE ADAyyyyy.WORK,LINK=DDWORKR1,SHARUPD=YES
/FILE ADAyyyyy.PLOGR1,LINK=DDPLOGR1,SHARUPD=YES
/FILE ADAyyyyy.PLOGR2,LINK=DDPLOGR2,SHARUPD=YES
/FILE ADAyyyyy.FILEA,LINK=DDFILEA,DEVICE=T C1,VOLUME=ADA001
```

```
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADACNV,DB=yyyyy,IDTNAME=ADABAS5B,TAPEREL=NO
ADACNV CONVERT TOVERS=vr
/STEP
/REMARK * NOW RELEASE THE TAPE
/REL DDFILEA,UNLOAD
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1	disk	
Multiple protection logs	DDPLOGRn	disk	
Intermediate storage	DDFILEA	tape/disk	
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADACNV parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADACNV messages	DDDRUCK	printer	*Messages and Codes*

ADACNV JCL Example (OS/390 or z/OS)

Refer to ADACNV in the MVSJOBS dataset for this example.

```
//ADACNV    JOB
//*
//*    ADACNV:
//*       EXAMPLE HOW TO USE ADACNV TO CONVERT DATABASE
//*       TO A DIFFERENT VERSION
//*
//CNV       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDFILEA   DD   DSN=EXAMPLE.DBYYYYY.FILEA,          <=== INTERMEDIATE
             UNIT=TAPE,VOL=SER=XXXXXX,DISP=(,KEEP)   <=== FILE
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADACNV,SVC=xxx,DE=3390,DBID=yyyyy
/*
//DDKARTE   DD   *
ADACNV CONVERT TOVERS=vr
/*
//
```

VM/ESA or z/MV

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKRn	disk	
Multiple protection logs	DDPLOGRn	disk	
Intermediate storage	DDFILEA	tape/disk	
ADARUN parameters	DDCARD	disk/terminal/reader	*Operations Manual*
ADACNV parameters	DDKARTE	disk/terminal/reader	
ADARUN messages	DDPRINT	disk/terminal/printer	*Messages and Codes*
ADACNV messages	DDDRUCK	disk/terminal/printer	*Messages and Codes*

ADACNV JCL Example (VM/ESA or z/VM)

```
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDPLOGR1,DSN=ADABASVv.PLOGR1,VOL=PLOGV
DATADEF DDPLOGR2,DSN=ADABASVv.PLOGR2,VOL=PLOGV
DATADEF DDFILEA,DSN=ADACNV.FILEA,MODE=A
DATADEF DDPRINT,DSN=ADACNV,DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADACNV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNCNV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=CONVERT.CONTROL,MODE=A
ADARUN
```

Contents of RUNCNV CONTROL A1:

```
ADARUN PROG=ADACNV,DEVICE=dddd,DB=yyyyy
```

Contents of CONVERT CONTROL A1:

```
ADACNV CONVERT TOVERS=vr
```

VSE/ESA

File	File Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
Work	WORKRn	disk	*	
Multiple protection logs	PLOGRn	disk	*	
Intermediate storage	FILEA	tape disk	SYS015 *	
ADARUN parameters	– CARD CARD	reader tape disk	SYSRDR SYS000 *	*Operations Manual*
ADACNV parameters	–	reader	SYSIPT	
ADARUN messages	–	printer	SYSLST	*Messages and Codes*
ADACNV messages	–	printer	SYS009	*Messages and Codes*

* Any programmer logical unit may be used.

ADACNV JCS Example (VSE/ESA)

See appendix B for a description of the VSE procedures.

Refer to member ADACNV.X for this example.

```
* $$ JOB JNM=ADACNV,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       CONVERT DATABASE TO NEW VERSION
// JOB ADACNV
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// DLBL FILEA,'ADACNV.WORK.FILE',0,SD
// EXTENT SYS015,,,,ssss,nnnn
// ASSGN SYS015,DISK,VOL=vvvvvv,SHR
// EXEC ADARUN,SIZE=ADARUN
ADARUN DBID=yyyyy,DEVICE=dddd,PROG=ADACNV,SVC=xxx
/*
ADACNV CONVERT TOVERS=vr
/*
/&
* $$ EOJ
```

5

ADADBS : DATABASE SERVICES

Functional Overview

The ADADBS utility performs the following functions:

Function	Action	Page
ADD	add a dataset to the Associator, Data Storage, protection log, or command log	139
ALLOCATE	allocate logical extents	141
CHANGE	change standard field length	143
CVOLSER	list logical extents on a given disk volume	145
DEALLOCATE	deallocate logical extent	146
DECREASE	decrease size of an existing Associator or Data Storage dataset	148
DELCP	delete checkpoint records	149
DELETE	delete file, protection log dataset, or command log dataset	151
DSREUSE	reuse Data Storage blocks	153
ENCODEF	change file encoding	155
INCREASE	increase size of an existing Associator or Data Storage dataset	156
ISNREUSE	reuse ISNs	164
MODFCB	modify file parameters – block padding factors – maximum blocks per extent – maximum compressed record length	165
NEWFIELD	add new field	168
ONLINVERT	start an online invert process	171
ONLREORFASSO	start an online reorder process for a file's Associator	173
ONLREORFDATA	start an online reorder process for a file's Data Storage	175
ONLREORFILE	start an online reorder process for a file's Associator and Data Storage	178
OPERCOM	issue Adabas operator commands	181
PRIORITY	change user priority	198
RECOVER	recover allocated space	199
REFRESH	set file to empty status	200

Function	Action	Page
REFRESHSTATS	refresh statistical values	201
RELEASE	release descriptor	203
RENAME	change file or database name	205
RENUMBER	change file number	206
RESETDIB	reset entry in active utility list	207
TRANSACTIONS	suspend/resume normal update transaction processing	209
UNCOUPLE	uncouple files	212

Note:
All ADADBS functions can also be performed using Adabas Online System (AOS).

Any number of functions may be performed during a single execution of ADADBS.

Syntax Checking with the TEST Parameter

The ADADBS functions now include a syntax-checking-only mode. When the TEST parameter is specified, the actual ADADBS function is checked, but not performed.

The ADADBS utility can perform multiple functions. As a result, ADADBS reads the parameters up to the next specified ADADBS function, and then executes the function/parameters just read. Then, ADADBS reads the function and parameters up to the following function, and so on. Therefore, to ensure that no functions are executed, the TEST parameter must be specified either before or within the first function/parameter group, as the following example shows:

```
ADADBS TEST
ADADBS DELETE FILE=1
ADADBS DELETE FILE=2
```

ADD : Add Dataset

The ADD function adds a new dataset to the Associator or Data Storage.

ADADBS ADD { **ASSOSIZE=**size [**ASSODEV=**{device-type | ADARUN-device}] |
DATASIZE=size [**DATADEV=**{device-type | ADARUN-device}] |
[**NOUSERABEND**]
[**TEST**]

Associator or Data Storage Dataset

For the Associator or for Data Storage, the dataset to be added may be on the same device type as that currently being used or on a different one. A maximum of five datasets each may be assigned to the Associator and Data Storage.

Note:
*The Associator and Data Storage dataset sizes must be added separately. It is **not** possible to add both with a single operation.*

After an ADD operation is completed for an Associator or Data Storage dataset, the ADD function automatically ends the current nucleus session. This allows for the necessary Associator or Data Storage formatting with ADAFRM before a new session is started. A message tells you that the nucleus has been stopped.

Procedure

To add an additional dataset to the Associator or Data Storage

1. Execute the ADD function.
2. Allocate the dataset with the operating system, then format the additional space using the ADAFRM utility.
3. Add necessary JCL/JCS to all Adabas nucleus and Adabas utility execution procedures.

Essential Parameter and Subparameter

ASSODEV | DATADEV : Device Type

The device type to be used for the new dataset. These parameters are required only if a different device type from the device type specified by the ADARUN DEVICE parameter is to be used.

For VSAM datasets, use dynamic device types; that is, DDxxxxR1=9999, DDxxxxR2=8888, ... DDxxxxR5=5555. For example, if DDDATAR3 is added, DATADEV=7777.

ASSOSIZE | DATASIZE : Size of Dataset to be Added

The number of cylinders to be contained in the new dataset.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Note that the validity of values and variables **cannot** be tested: only the syntax of the specified parameters can be tested. See page 138 for more information about using the TEST parameter in ADADBS functions.

Examples

A new dataset containing 800 cylinders on 3350 disks is to be added to Data Storage.

```
ADADBS ADD DATASIZE=800, DATADEV=3350
```

A new dataset containing 100 cylinders is to be added to the Associator on the Associator's existing device type.

```
ADADBS ADD ASSOSIZE=100
```

ALLOCATE : Allocate File Extent

The ALLOCATE function may be used to allocate an address converter, Data Storage, normal or upper index extent of a specific size. Only one extent may be allocated per ADADBS execution.

ADADBS ALLOCATE **FILE=**file-number
{ **ACSIZE** | **DSSIZE** | **NISIZE** | **UISIZE** }**=**size
[**DEVICE=**{device-type | ADARUN-device}]
[**NOUSERABEND**]
[**PASSWORD='**password**'**]
[**STARTRABN=**start-rabn]
[**TEST**]

Essential Parameters

FILE : File for Which an Extent Is Allocated

FILE specifies the number of the file for which the extent is to be allocated.

ACSIZE | DSSIZE | NISIZE | UISIZE : Extent Type and Size

These parameters are used to indicate the type and size of the extent to be allocated. One and only one extent type and size can be specified in a single ADADBS ALLOCATE statement. The specified value can be either cylinders or blocks; a size in blocks must be followed by "B" (for example, 2000B).

Optional Parameters

DEVICE : Device Type

The device type to be used for file allocation. This parameter is required only if a different device type from the device type specified by the ADARUN DEVICE parameter is to be used.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

The password of the file. This parameter is required if the file is password-protected.

STARTRABN : Starting RABN for Extent

The beginning RABN of the extent to be allocated. If this parameter is omitted, ADADBS will assign the starting RABN.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

An address converter extent of 30 blocks is to be allocated for file 15.

```
ADADBS ALLOCATE FILE=15,ACSIZE=30B
```

CHANGE : Change Standard Length of a Field

The CHANGE function can be used to change

- the standard length of an Adabas field;
- a normal alphanumeric (A) field to a long-alpha (LA) field; or
- the default field format from unpacked (U) to packed (P).

Only one of these changes may be performed per function execution.

No modifications to records in Data Storage are made by this function. The user is, therefore, responsible for preventing references to the field that would cause invalid results because of an inconsistency between the new standard length as defined to Adabas and the actual number of bytes contained in the record.

When changing the length of an Adabas expanded file field, the change must be made to **each individual component file** of the expanded file. Each CHANGE operation on a component file causes a message that confirms the change, and returns condition code 4.

```
ADADBS CHANGE  FILE=file-number
               FIELD='field-name
               { FORMAT=P | LENGTH=new-length | OPTION=LA }
               [NOUSERABEND]
               [PASSWORD='password']
               [TEST]
```

Essential Parameters

FILE : File Containing the Field

The file in which the field whose length is to be changed is contained. An Adabas system file may not be specified.

FIELD : Field to be Changed

The field whose standard length is to be changed. The field cannot be one that was defined with the FI option, or a field with a defined length of zero (variable-length field). Specify the field name between apostrophes (').

FORMAT=P : New Field Format

The new standard field format. The only field format change supported is from ‘U’ (unpacked) to ‘P’ (packed). The field cannot be parent of a sub-/super-/hyperdescriptor.

One of the parameters FORMAT, LENGTH, or OPTION must be specified; but only one of the three may be specified.

LENGTH : New Field Length

The new standard length for the field. A length of 0 is not permitted, nor can a field with an existing defined length of zero (such as a variable-length field) be redefined to a standard length.

One of the parameters FORMAT, LENGTH, or OPTION must be specified; but only one of the three may be specified.

OPTION=LA : New Field Option

The new field option. The only field option change supported is from normal alphanumeric (A) to long-alpha (LA).

One of the parameters FORMAT, LENGTH, or OPTION must be specified; but only one of the three may be specified.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

The password of the file containing the field to be changed. This parameter is required if the file is password-protected.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

The standard length of field AB in file 5 is to be changed to 11 bytes.

```
ADADBS CHANGE FILE=5,FIELD='AB',LENGTH=11
```

CVOLSER : Print Adabas Extents on Given Volume

The CVOLSER function is used to print the Adabas file extents contained on a disk volume.

ADADBS CVOLSER **VOLSER=**volume-serial-number
[**NOUSERABEND**]
[**TEST**]

Essential Parameter

VOLSER : Volume Serial Number

VOLSER is the volume serial number of the disk volume to be used.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

The Adabas file extents contained on disk volume DISK02 are to be printed.

```
ADADBS CVOLSER VOLSER=DISK02
```

DEALLOCATE : Deallocate File Extent

The DEALLOCATE function may be used to deallocate an address converter, Data Storage, normal index or upper index extent. Only one extent may be deallocated per ADADBS execution.

```
ADADBS DEALLOCATE  FILE=file-number
                   { ACSIZE | DSSIZE | NISIZE | UISIZE }=size
                   [NOUSERABEND]
                   [PASSWORD='password']
                   [STARTRABN=start-rabn]
                   [TEST]
```

Essential Parameters

ACSIZE | DSSIZE | NISIZE | UISIZE : Extent Type and Size

These parameters specify the type and size of extent to be deallocated. One and only one extent type and size may be specified. The size must be in number of RABN blocks followed by "B" (for example, DSSIZE=20B), and cannot exceed the number of unused RABNs at the end of an extent.

FILE : File for Which an Extent Is Deallocated

FILE specifies the file for which the extent is to be deallocated. Specify a decimal value.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

The password of the file for which space is to be deallocated. This parameter is required if the file is password-protected. Specify the password between apostrophes (').

STARTRABN : Starting RABN for Extent

The first RABN of the extent in which deallocation is to take place. If this parameter is omitted, the last extent for the file will be deallocated. In the address converter, only the last extent may be deallocated.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

An address converter extent of 30 blocks is to be deallocated for file 15.

```
ADADBS DEALLOCATE FILE=15,ACSIZE=30B
```

DECREASE : Decrease Associator/Data Storage

The DECREASE function decreases the size of the last dataset currently being used for Associator or Data Storage. The space to be released must be available in the free space table (FST).

The DECREASE function does **not** deallocate any of the specified physical extent space.

ADADBS DECREASE { **ASSOSIZE** | **DATASIZE** }=size**B**
[**NOUSERABEND**]
[**TEST**]

Essential Parameter

ASSOSIZE | DATASIZE : Blocks to Be Decreased

ASSOSIZE/DATASIZE define the number of blocks by which the Associator or Data Storage dataset is to be decreased, specified as a decimal value followed by “B”. Either ASSOSIZE or DATASIZE can be specified, but not both. If both ASSOSIZE and DATASIZE are to be specified, each must be entered on a separate ADADBS DECREASE statement.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

The Associator is to be decreased by 100 blocks and Data Storage is to be decreased by 200 blocks.

```
ADADBS DECREASE ASSOSIZE=100B
ADADBS DECREASE DATASIZE=200B
```

Procedure

To deallocate space, perform the following steps:

1. Decrease the database with the DECREASE function;
2. Save the database with ADASAV SAVE;
3. Reformat the datasets with ADAFRM;
4. Restore the database with ADASAV.

DELCP : Delete Checkpoint Records

The DELCP function deletes checkpoint records.

After running ADADBS DELCP, the remaining records are reassigned ISNs to include those ISNs made available when the checkpoint records were deleted. The lower ISNs are assigned but the chronological order of checkpoints is maintained.

```
ADADBS  DELCP   TODATE=yyyymmdd
                [NOUSERABEND]
                [TEST]
```

Essential Parameter

TODATE : Last Date for Deleted Records

TODATE specifies the latest date for which checkpoint information is deleted. Checkpoint information dated after the date specified by TODATE= is not deleted. TODATE= must be specified; there is no default date. Specify the date as a four-digit decimal value for year followed by two-digit decimal values for month and day, in that order.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

All checkpoint records up to and including February 1, 1996 are to be deleted.

```
ADADBS DELCP TODATE=19960201
```

DELETE : Delete File

The DELETE function deletes an Adabas file from the database.

ADADBS DELETE {**FILE=**fnr [**KEEPFDT**][**PASSWORD=**'password']}
[**NOUSERABEND**]
[**TEST**]

When an Adabas file is deleted from the database, all logical extents assigned to the file are deallocated. The released space may be used for a new file or for a new extent of an existing file.

The file to be deleted may not be coupled. If an Adabas expanded file is specified, the complete expanded file (the anchor and all component files) is deleted.

When the DELETE function completes successfully, any locks previously set with the operator commands LOCKU or LOCKF are reset.

Essential Parameter

FILE : File to Be Deleted

FILE specifies the number of the Adabas file to be deleted. An Adabas system file may be specified **only** if ADADBS DELETE is the only Adabas user; deleting a system file automatically causes Adabas to terminate when finished. Adabas system files are checkpoint, security, triggers, and any other files loaded with the ADALOD utility's SYSFILE option. To delete an Adabas expanded file, specify the file number (also the anchor file).

Optional Parameters

KEEPFDT : Retain the Field Definition Table

The KEEPFDT parameter, if specified, instructs ADADBS DELETE to keep the deleted file's field definition table (FDT) for later use by ADACMP. If this parameter is specified, a file with the same number as the one now being deleted can only be later loaded if either the new file's FDT is the same as that of the deleted file, or the load operation specifies the IGNFDT parameter to accept the new file's FDT.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

PASSWORD specifies the password of the file to be deleted. This parameter is required if the file is password-protected.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Examples

File 6 is to be deleted.

```
ADADBS DELETE FILE=6
```

Password-protected file 10 is to be deleted. The field definition table is to be retained. File number 10 cannot be used again until another ADALOD LOAD command is issued with the IGNFDT option.

```
ADADBS DELETE FILE=10,KEEPFDT,PASSWORD='FILE10'
```

DSREUSE : Reuse Data Storage Blocks

The DSREUSE function controls the assignment of Data Storage blocks.

```
ADADBS  DSREUSE     FILE=file-number
                    MODE={ ON | OFF }
                    [NOUSERABEND]
                    [PASSWORD='password']
                    [RESET]
                    [TEST]
```

Essential Parameters

FILE : File Number

FILE is the number of the file for which the DSREUSE setting is to apply.

Block reuse is originally determined when the file is loaded into the database with the ADALOD FILE function, or when the system file is defined with the ADADEF DEFINE function. In both cases, block reuse defaults to “YES” unless specified otherwise in those functions.

MODE : Reuse Mode

The Data Storage block assignment mode to be in effect. MODE=OFF indicates that Data Storage blocks which become free as a result of record deletion may not be reused, in effect cancelling the ADADBS DSREUSE function. MODE=ON indicates that Data Storage blocks may be reused. The MODE= parameter has no default, and must be specified.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

PASSWORD specifies the file's security password, and is required if the file is password-protected.

RESET : Reset Space Pointer

The RESET parameter causes searches for new Data Storage space to start at the beginning of the file.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

Data Storage blocks for file 6 are not to be reused.

```
ADADBS DSREUSE FILE=6,MODE=OFF
```

ENCODEF : Change File Encoding

```
ADADBS ENCODEF      FILE=file-number
                    [ FACODE=alpha-key ]
                    [ UWCODE=wide-key ]
                    [ NOUSERABEND ]
                    [ TEST ]
```

Essential Parameter

FILE : File Number

FILE is the number of the file for which encoding is to be changed.

Optional Parameters

FACODE : Encoding for Alphanumeric Fields in File

The FACODE parameter defines the encoding for alphanumeric fields stored in the file. It can be applied to files already loaded. The encoding must be derived from EBCDIC encoding; that is, X'40' is the space character. Double-byte character set (DBCS) type encodings are supported with the exception of DBCS-only. See appendix C for a list of supplied code pages.

FACODE and/or UWCODE must be specified.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information about using the TEST parameter in ADADBS functions.

UWCODE : User Encoding for Wide-Character Fields in File

The UWCODE parameter defines the user encoding for wide-character fields stored in the file. It can be applied to files already loaded. Note that the wide file encoding is not changed.

To change the encoding of wide-character fields, the file must be unloaded, decompressed, compressed, and reloaded.

FACODE and/or UWCODE must be specified. See appendix C for a list of supplied code pages.

Example

In file 1425, change the encoding of alphanumeric fields to use code page 285 (CECP: United Kingdom, EBCDIC-compatible with X‘40’ fill character) and change the encoding of wide fields to use code page 3396 (IBM, CCSID 4396, Japanese host double byte including 1880 user-defined characters). Note that because UWCODE is changing, the file must be unloaded, decompressed, compressed, and reloaded.

```
ADADBS ENCODEF FILE=1425,FACODE=285,UWCODE=3396
```

INCREASE : Increase Associator/Data Storage

The INCREASE function increases the size of the last dataset currently being used for the Associator or Data Storage. This function may be executed any number of times for the Associator. The maximum of five Data Storage space tables (DSSTs) limits Data Storage increases to four before all five Data Storage extents must be combined into a single extent with either the REORASSO or REORDB function of the ADAORD utility.

Notes:

1. *The Associator and Data Storage dataset sizes must be increased separately. It is **not** possible to increase both with a single operation.*
2. *After an INCREASE operation is completed, the INCREASE function automatically ends the current nucleus session. This allows for the necessary Associator or Data Storage formatting with ADAFRM before a new session is started. An informational message occurs to tell you that the nucleus has been stopped.*

```
ADADBS INCREASE     {ASSOSIZE | DATASIZE }=size
                    [ NOUSERABEND ]
                    [ TEST ]
```

Essential Parameter

ASSOSIZE | DATASIZE : Size to Be Increased

The additional number of blocks or cylinders needed by the Associator or Data Storage dataset. To specify blocks, add "B" after the value; for example, DATASIZE=50B.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

Use the TEST parameter to test the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information about using this parameter.

Example

The Associator is to be increased by 400 cylinders.

```
ADADBS INCREASE ASSOSIZE=400
```

General Procedure

The general procedure for increasing the size of the Associator or Data Storage is as follows:

1. Back up the database using the ADASAV utility. This step is optional but recommended.
2. Execute the ADADBS INCREASE function.

Note:
After an INCREASE operation is completed, the INCREASE function automatically ends the current nucleus session. This allows for the necessary Associator or Data Storage formatting with ADAFRM before a new session is started. An informational message occurs to tell you that the nucleus has been stopped.

3. Format the additional space being added to the dataset with the ADAFRM utility.

Operating-System-Specific Procedures

OS/390 or z/OS Systems

Under OS/390 or z/OS, the same dataset may be formatted by specifying the DISP=MOD parameter in the JCL. The SPACE parameter for the dataset being increased should be set to

SPACE=(CYL,(0,*n***))**

—where "n" is the amount of space (in cylinders) being added. The ADAFRM control statement should also specify the number of cylinders being added. If the increased part of the dataset to be formatted is contained on a new volume, the VOL parameter of the JCL must include references to all volumes containing the dataset.

Example 1: OS Single-Volume INCREASE

400 cylinders are to be added to an Associator dataset which currently contains 300 cylinders. The control statement for the INCREASE function would be

```
ADADBS INCREASE ASSOSIZE=400
```

The following JCL example increases the Associator dataset using ADAFRM:

```
//DDASSOR1  DD  DSN=....,DISP=MOD,SPACE=(CYL,(0,400))
```

—and the actual ADAFRM control statement would be

```
ADAFRM ASSOFRM SIZE=400
```

Example 2: OS Multivolume INCREASE

To provide the increase in example 1 for multiple volumes, specify the volumes in the JCS:

```
//DDASSOR1  DD  DSN=...
//              DISP=(MOD,CATLG),VOL=SER=(V1,V2,...),SPACE=(CYL,(0,400))...
```

Include the following step after the INCREASE step but before the FORMAT step to ensure a correct catalog entry:

```
//UNCATLG  EXEC PGM=IEFBR14
//DDASSOR1  DD  DSN=...,DISP=(SHR,UNCATLG)
```

VSE/ESA Systems

The following procedures are recommended for increasing Associator or Data storage:

1. Save the current database.
2. Update the JCS defining the database to add the new extent on the same volume.

Before a new Associator or Data extent **on either a different or the same VSE volume** can be increased with ADADBS INCREASE and formatted with ADAFRM, that volume's table of contents (VTOC) must be updated to contain the new extent.

Use a job similar to the following example to update the VTOC for a single volume extent:

```
* $$ JOB JNM=jobname
* $$ LST ...
* $$ PCH ...
// ASSGN SYS001,DISK,VOL=volume,SHR
// DLBL ASSOEXT,'dsname',99/365,DA
// EXTENT SYS001,volume1,1,0,starttrack1,trackcount1
// EXTENT SYS001,volume1,1,1,starttrack2,trackcount2
```

```
// EXEC ASSEMBLY,GO
MODVTOC  CSECT
         BALR 9,0
         BCTR 9,0
         BCTR 9,0
         USING MODVTOC,9
         OPEN  ASSOEXT
         CLOSE ASSOEXT
         EOJ   RC=0
ASSOEXT  DTFPH TYPEFLE=OUTPUT,DEVADDR=SYS001,DEVICE=DISK,MOUNTED=ALL
         END
/*
/&
* $$ EOJ
```

For a two-volume extent, use a job similar to the following example:

```
* $$ JOB JNM=jobname
* $$ LST ...
* $$ PCH ...
// ASSGN SYS001,DISK,VOL=volume1,SHR
// ASSGN SYS002,DISK,VOL=volume2,SHR
// DLBL ASSOEXT,'dsname',99/365,DA
// EXTENT SYS001,volume1,1,0,starttrack1,trackcount1
// EXTENT SYS002,volume2,1,1,starttrack2,trackcount2
// EXEC ASSEMBLY,GO
MODVTOC  CSECT
         BALR 9,0
         BCTR 9,0
         BCTR 9,0
         USING MODVTOC,9
         OPEN  ASSOEXT
         CLOSE ASSOEXT
         EOJ   RC=0
ASSOEXT  DTFPH TYPEFLE=OUTPUT,DEVADDR=SYS001,DEVICE=DISK,MOUNTED=ALL
         END
/*
/&
* $$ EOJ
```

Note:
This job causes VSE error message 4733D to be sent to the console, and the operator is asked for a response. After the JCS has been validated, the operator response should be "DELETE".

3. Perform the ADADBS INCREASE operation.

4. Run the new ADAFRM job to format the new extent. The ADAFRM job must specify the FROMRABN parameter, as shown in the following example:

   ADAFRM ASSOFRM SIZE=size**,FROMRABN=**rabn-number

   —where “size” is the number of cylinders or blocks by which the dataset is to be increased, and “rabn-number” is the first RABN in the new extent.

5. Start the Adabas nucleus.

   Note:
   In a VM environment, certain restrictions apply to multivolume, multiextent files. If these restrictions are violated, VSE error 4n83I (invalid logical unit) may occur. Refer to the appropriate IBM documentation for more information about these restrictions.

VM/ESA or z/VM Systems

Under VM/ESA or z/VM, there are two procedures for increasing the database. The first uses the ADAMAINT and INCREASE EXECs; the second is a step-by-step manual procedure.

EXEC Procedure

1. Call the ADAMAINT EXEC to modify your CMS environment:

   ADFnnnnn **EXEC, DB**nnnnn **VOLUMES** volume**=**vol-id**,** ...

   ADAMAINT lets you add a new minidisk to an existing ASSO/DATA/WORKRx, or define a new ASSO/DATA/WORKRx.

2. Call the INCREASE EXEC. This EXEC automatically does a LINK, an ADADBS ADD, or an ADADBS INCREASE (depending on what you specify in ADAMAINT), followed by an ADAFRM to format the new area.

Manual Procedure

1. Define a new minidisk that is one cylinder (or pseudo-cylinder) larger than the required size.

2. Issue the FORMAT command

   FORMAT cuu **T** nnn

   —where “cuu” is the virtual unit address of the new minidisk and “nnn” is “1” for a CKD device or “20” for an FBA device. When prompted for a volume label, you must specify a unique name of up to six alphanumeric characters.

3. Reserve the minidisk with the following command:

   RESERVE file-name file-type **T**

   —where “file-name” and “file-type” match the file name and file type used for the file on the primary minidisk.

4. Execute the ADADBS INCREASE utility as described in this chapter.

5. Produce an ADAREP report, and find the first RABN in the new extent. This may be located in the physical layout of the database. The RABN range on this extent indicates VOLSER NUMBER “xxxxxx”.

6. Add CP LINK statements for the new minidisk to the directory or PROFILE EXEC, as required. Update any PROFILE EXECs or CP directory entries for any other virtual machines with multiwrite access to this database (for example, the DBA machine).

7. For any EXECs that require it, modify the DATADEF statements for the file. If the standard Software AG EXECs are being used, these DATADEF statements are found in the ADFnnnnn EXEC, where “nnnnn” is the five-digit database ID.

   To modify the DATADEF statement, locate the line

   volx = vol-id

   —where “volx” is “a” for the Associator or “d” for DATA, and “vol-id” is the previous volume list. Change this line to

   volx = (**vol-id**,**vo-label**)

   —where “vo-label” is the volume label specified while entering the FORMAT command in step 2.

8. Execute the ADAFRM utility for the file as

   ADAFRM xxxx**FRM SIZE=**size**,FROMRABN=**rabn-number

   —where

xxxx	is either ASSO or DATA
size	is the size of the minidisk minus one cylinder (or psuedo-cylinder)
rabn-number	is the first RABN on the new extent as shown in the report created in step 6.

BS2000 Systems

Use the following procedure to increase the database on BS2000 systems:

1. Execute ADADBS INCREASE as described on page 157.
2. Produce a database report by running the ADAREP utility. Use the report to find the first RABN for the new extent in the “Physical Layout of the Database” portion of the report. The RABN range is indicated in the “VOLSER NUMBER” column.
3. Increase the dataset with the BS2000 “MODIFY–FILE–ATTRIBUTE” command.
   For example:

   /MODIFY–FILE–ATTRIBUTE ADA99.ASSO,PUB(SPACE=REL(400))

   Note:
   In the old ISP format, this was performed by the “FILE” command; for example, /FILE ADA99.ASSO, SPACE=400.

4. Format the new space by running the ADAFRM utility. An example for the space added in step 4 is

   ADAFRM ASSOFRM SIZE=400B,FROMRABN=rabn-number

   —where “rabn-number” specifies the first RABN shown on the new extent, as shown in the report.

ISNREUSE : Reuse ISNs

The ISNREUSE function controls whether ISNs of deleted records may be reassigned to new records.

```
ADADBS  ISNREUSE    FILE=file-number
                    MODE={ ON | OFF }
                    [NOUSERABEND]
                    [PASSWORD='password']
                    [RESET]
                    [TEST]
```

Essential Parameters

FILE : File Number

FILE is the number of the file for which the ISNREUSE setting is to be changed. The checkpoint file cannot be specified.

MODE : Reuse Mode

MODE causes the ISN reuse mode to be in effect. MODE=OFF causes Adabas not to reuse the ISN of a deleted record for a new record. Each new record will be assigned the next higher unused ISN. MODE=ON indicates that Adabas may reuse the ISN of a deleted record. The MODE parameter has no default; it must be specified.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

PASSWORD specifies the file's security password, and is required if the file is password-protected.

RESET : Reset ISN Pointer

The RESET parameter causes searches for an unused ISN to start at the beginning of the file.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Note that the validity of values and variables **cannot** be tested: only the syntax of the specified parameters can be tested. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

ISNs of deleted records in file 7 may be reassigned to new records.

```
ADADBS ISNREUSE FILE=7,MODE=ON
```

MODFCB : Modify File Parameters

The MODFCB function modifies various parameters for a non-system Adabas file.

ADADBS MODFCB **FILE=**file-number
[**ASSOPFAC=**new-padding-factor]
[**DATAPFAC=**new-padding-factor]
[**MAXDS=**maximum-secondary-allocation]
[**MAXNI=**maximum-secondary-allocation]
[**MAXUI=**maximum-secondary-allocation]
[**MAXRECL=**maximum-compressed-record-length]
[**NOUSERABEND**]
[**PASSWORD=**'password']
[**PGMREFRESH**={**YES** | **NO** }]
[**TEST**]

Essential Parameter

FILE : File Number

FILE is the number of the Adabas file to be modified. An Adabas system file cannot be specified.

Optional Parameters

ASSOPFAC / DATAPFAC : File Padding Factors

ASSOPFAC/DATAPFAC specify the padding factor (1–90) to be in effect for Associator and Data Storage, respectively. Existing blocks retain their original padding factor (see the ADAORD utility).

MAXDS / MAXNI / MAXUI : Maximum Secondary Allocation

The maximum number of blocks per secondary extent allocation for the Data Storage (MAXDS), the normal index (MAXNI), and the upper index (MAXUI).

The value specified must specify blocks, be followed by “B” (for example, MAXDS=8000B), and cannot be more than 65535B.

If one of the parameters is either not specified or specifies “0B”, the maximum secondary extent allocation for that component has no limit.

In all cases, however, Adabas enforces minimum secondary allocations for these parameters:

MAXDS=6B
MAXNI=6B
MAXUI=15B

If you specify a value lower than these minimum allocations, the minimum value is used.

MAXRECL : Maximum Compressed Record Length

The maximum compressed record length permitted for the file. The value specified should not be less than the current maximum record size in the specified file.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

This parameter is required if the file is password-protected.

PGMREFRESH : Program-generated File Refresh

The PGMREFRESH option determines whether a user program is allowed to perform a file refresh operation by issuing a special E1 command. If the parameter is not specified, the option remains in its current status: either on (YES) or off (NO).

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

The following modifications are to be made for file 203: the Associator padding factor is set to 5, the Data Storage padding factor to 5, and the maximum Data Storage secondary extent allocation to 100 blocks.

```
ADADBS MODFCB FILE=203,ASSOPFAC=5,DATAPFAC=5,MAXDS=100B
```

NEWFIELD : Add New Field

The NEWFIELD function adds one or more fields to a file. The new field definition is added to the end of the field definition table (FDT).

Note:
Although the definition of a descriptor field is independent of the record structure, note that if a descriptor field is not ordered first in a record and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to order the field first for each record of the file.

NEWFIELD cannot be used to specify actual Data Storage data for the new field; the data can be specified later using Adabas add/update or Natural commands.

When adding a field to an Adabas expanded file, the field must be added to **each individual component file**. Each NEWFIELD operation on a component file returns a message that confirms the change and condition code 4.

ADADBS NEWFIELD **FILE=**file-number
[**FNDEF=**'Adabas-field-definition']
[**NOUSERABEND**]
[**PASSWORD=**'password']
[**SUBFN=**'name=parent-field(begin,end)']
[**SUPFN=**'name={parent-field(begin,end)},...']
[**TEST**]

Essential Parameter

FILE : File Number

FILE specifies the file in which the field to be added is contained. The file may not be an Adabas system file.

Optional Parameters

FNDEF : Adabas Field Definition

FNDEF specifies an Adabas field (data) definition. One FNDEF statement is required for each field to be added. The syntax used in constructing field definition entries is

FNDEF='level**,**name[**,**length**,**format][**,**option,...] '

Each definition must adhere to the field definition syntax as described for the ADACMP utility starting on page 66.

Note the following restrictions:

- A subdescriptor, superdescriptor, hyperdescriptor, or phonetic descriptor definition cannot be specified.
- Text information or sequence numbers are not permitted.
- If you specify an occurrence number when adding an MU or PE field, it is ignored.

The following rules apply when you set the level number in the first FNDEF statement:

1. A level number 01 is always allowed.
2. A level number of 02 or higher means that this field is to be added to an existing group. If so, the following rules apply:

   - The field can be added if the group is a normal (not periodic) group;
   - If the group is a PE, the field can be added only if the file control block (FCB) for the file does **not** exist; that is, either the file was deleted with the KEEPFDT option, or the FDT was defined using the Adabas Online System “Define FDT” function but the “Define File” function has not yet been run.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

File password. This parameter is required if the file is password-protected.

SUBFN / SUPFN : Add Subfields or Superfields

These parameters may be used to add subfields and superfields. Each definition must adhere to the definition syntax for sub/superfields as described for the ADACMP utility.

TEST : Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

Group AB (consisting of fields AC and AX) is to be added to file 24.

```
ADADBS  NEWFIELD   FILE=24
ADADBS             FNDEF='01,AB'
ADADBS             FNDEF='02,AC,3,A,DE,NU'
ADADBS             FNDEF='02,AX,5,P,NU'
```

ONLINVERT : Start Online Invert

The ONLINVERT function starts an online invert process.

```
ADADBS ONLINVERT     FILE=file-number
                          { FIELD='field-name [,option,...]'  |
                          SUPDE='name [,UQ [,XI] ]'=parent-field (begin,end),...'  |
                          SUBDE='name [,UQ [,XI] ]'=parent-field (begin,end)' |
                          PHONDE='phonde-name (parent-field)' |
                          HYPDE='nr,name,length,format [,options]=parent-field,...' |
                          COLDE='nr,name [,UQ [,XI] ]'=parent-field' }
                     [CODE=cipher-code]
                     [PASSWORD=password]
                     [NOUSERABEND]
                     [TEST]
                     [WAIT]
```

Essential Parameters

FILE : File Number

File is the number of the file for which the new descriptor is to be created. If a component file of an expanded file chain is specified, the descriptor is added to all component files of that chain.

FIELD / SUBDE / SUPDE / PHONDE / HYPDE / COLDE : Define Descriptor

Exactly one of these parameters must be used to define the type of descriptor to be inverted. Only one descriptor per file can be inverted at a time using the online invert function.

Use the FIELD parameter to define a field as descriptor; use the COLDE parameter for a collation descriptor; the HYPDE parameter for a hyperdescriptor; PHONDE for a phonetic descriptor; SUBDE for a subdescriptor; and SUPERDE for a superdescriptor.

FIELD specifies an existing field to be inverted. The field may be an elementary or multiple-value field and may be contained within a periodic group (unless the field is defined with the FI option).

If the descriptor is to be unique, specify “UQ” following the field name. A field in a periodic group cannot be defined as a unique descriptor. If the uniqueness of the descriptor is to be determined with the index (occurrence number) excluded, specify “XI” as well.

When inverting a sub- or superfield, the respective SUBDE or SUPDE parameter must specify the same parent fields that were specified when the field was created; otherwise, an error occurs. Begin and end values are taken from the original field definitions.

If a parent field with the NU option is specified, no entries are made in the inverted list for those records containing a null value for the field. For super- and hyperdescriptors, this is true regardless of the presence or absence of values for other descriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

See the **ADACMP** utility description starting on page 43 for detailed information about the individual descriptor syntax, subparameter values, and coding.

Optional Parameters

CODE : Cipher Code

If the file specified with the FILE parameter is ciphered, an appropriate cipher code must be supplied using the CODE parameter.

PASSWORD : File Password

If the file specified with the FILE parameter is security-protected, the file's password must be supplied using the PASSWORD parameter.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

WAIT : Wait for End of Process

Specify WAIT if ADADBS is to wait for the end of the online process before proceeding either with the next function or with termination.

If WAIT is not specified, ADADBS proceeds immediately after initiating the online process.

Example

Initiate an online process to make field AA of file 10 a descriptor, without waiting for the end of this process.

```
ADADBS ONLINVERT FILE=10,FIELD=AA
```

ONLREORFASSO : Start Online Reorder Associator for Files

The ONLREORFASSO function starts an online process to reorder the Associator of the specified files.

Notes:

1. *The online reorder process does not change the existing file extents but only reorganizes the file's index within these extents.*
2. *The online index reorder process dows not move index elements out of blocks that are full (according to the Asso padding factor); it only moves elements into blocks that are not full.*
3. *Released index blocks are put into the unused RABN chain, which can be viewed using the ADAICK ICHECK utility function.*

```
ADADBS ONLREORFASSO   FILE=file-number
                      [ASSOPFAC=asso-padding-factor]
                      [PASSWORD=password]
                      [NOUSERABEND]
                      [TEST]
                      [WAIT]
```

Essential Parameters

FILE : File Number

FILE specifies the file to which the parameters that follow in the statement sequence apply.

Several files and their related parameters may be specified within one ONLREORFASSO operation. In this case, the files are reordered in the specified sequence.

If a component file of an Adabas expanded file is specified, only that file's Associator is reordered; this has no adverse effect on the other component files.

The Adabas checkpoint or security file number must not be specified.

Optional Parameters

ASSOPFAC : Associator Padding Factor

ASSOPFAC defines the Associator block padding factor, which is the percentage of each Associator block **not** used during the reorder process. Specify a value in the range 1–90. The number of bytes free after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current padding factor in effect for the file is used.

PASSWORD : File Password

If the file is password-protected, use this parameter to specify the password.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility does **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

WAIT : Wait for End of Process

Specify WAIT if ADADBS is to wait for the end of the online process before proceeding either with the next function or with termination.

If WAIT is not specified, ADADBS proceeds immediately after initiating the online process.

Example

Initiate an online process that reorders the Associator of file 10 first and then file 11. The Associator padding factor of file 11 is to be 5 percent.

```
ADADBS ONLREORFASSO FILE=10
ADADBS              FILE=11,ASSOPFAC=5
```

ONLREORFDATA : Start Online Reorder Data for Files

The ONLREORFDATA function starts an online process to reorder the Data Storage of the specified files.

Note:
The online reorder process does not change the existing file extents but only reorganizes the file's Data Storage records within these extents.

ADADBS ONLREORFDATA **FILE=**file-number
[**DATAPFAC=**data-padding-factor]
[**SORTSEQ=**{**ISN** | de-name | physical-sequence}]
[**PASSWORD=**password]
[**NOUSERABEND**]
[**TEST**]
[**WAIT**]

Essential Parameters

FILE : File Number

FILE specifies the file to which the parameters that follow in the statement sequence apply.

Several files and their related parameters may be specified within one ONLREORFDATA operation. In this case, the files are reordered in the specified sequence.

If a component file of an Adabas expanded file is specified, only that file's Data Storage is reordered; this has no adverse effect on the other component files.

The Adabas checkpoint or security file number must not be specified.

Optional Parameters

DATAPFAC : Data Storage Padding Factor

DATAPFAC specifies the Data Storage padding factor. The number specified represents the percentage of each Data Storage block that remains unused when the file is reordered. A value in the range 1–90 may be specified (see the **ADALOD** utility chapter starting on page 313 for additional information about setting and using the Data Storage padding factor).

If this parameter is omitted, the current padding factor in effect for the file is used.

SORTSEQ : File Reordering Sequence

SORTSEQ determines the sequence in which the file is processed. If this parameter is omitted, the records are processed in physical sequence.

Note:
Records within a single Data Storage block are not sorted according to the specified sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. **Do not** use a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

If ISN is specified, the file is processed in ascending ISN sequence.

PASSWORD : File Password

If the file is password-protected, use this parameter to specify the password.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility does **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

WAIT : Wait for End of Process

Specify WAIT if ADADBS is to wait for the end of the online process before proceeding either with the next function or with termination.

If WAIT is not specified, ADADBS proceeds immediately after initiating the online process.

Example

Initiate an online process that reorders the Data Storage of file 10 first, and then file 11. The Data Storage padding factor of file 11 is to be 5 percent.

```
ADADBS ONLREORFDATA FILE=10
ADADBS              FILE=11,DATAPFAC=5
```

ONLREORFILE : Start Online Reorder Associator and Data for Files

The ONLREORFILE function starts an online process to reorder the Associator and Data Storage of the specified files.

Note:
The online reorder process does not change the existing file extents but only reorganizes the file's index and Data Storage records within these extents.

ADADBS ONLREOR FILE **FILE=**file-number
[**ASSOPFAC=**asso-padding-factor]
[**DATAPFAC=**data-padding-factor]
[**SORTSEQ=**{**ISN** | de-name | physical-sequence}]
[**PASSWORD=**password]
[**NOUSERABEND**]
[**TEST**]
[**WAIT**]

Essential Parameters

FILE : File Number

FILE specifies the file to which the parameters that follow in the statement sequence apply.

Several files and their related parameters may be specified within one ONLREORFILE operation. In this case, the files are reordered in the specified sequence.

If a component file of an Adabas expanded file is specified, only that file's Associator and Data Storage is reordered; this has no adverse effect on the other component files.

The Adabas checkpoint or security file number must not be specified.

Optional Parameters

ASSOPFAC : Associator Padding Factor

ASSOPFAC defines the new Associator block padding factor, which is the percentage of each Associator block **not** used during the reorder process. Specify a value in the range 1–90. The number of bytes free after padding must be greater than the largest descriptor value plus 10.

If this parameter is omitted, the current padding factor in effect for the file is used.

DATAPFAC : Data Storage Padding Factor

DATAPFAC specifies the new Data Storage padding factor. The number specified represents the percentage of each Data Storage block that remains unused when the file is reordered. A value in the range 1–90 may be specified (see the **ADALOD** utility chapter starting on page 313 for additional information about setting and using the Data Storage padding factor).

If this parameter is omitted, the current padding factor in effect for the file is used.

SORTSEQ : File Reordering Sequence

SORTSEQ determines the sequence in which the file is processed. If this parameter is omitted, the records are processed in physical sequence.

Note:
Records within a single Data Storage block are not sorted according to the specified sequence.

If a descriptor is specified, the file is processed in the logical sequence of the descriptor values. **Do not** use a hyperdescriptor, a phonetic descriptor, a multiple-value field, or a descriptor contained in a periodic group.

If ISN is specified, the file is processed in ascending ISN sequence.

PASSWORD : File Password

If the file is password-protected, use this parameter to specify the password.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility does **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

WAIT : Wait for End of Process

Specify WAIT if ADADBS is to wait for the end of the online process before proceeding either with the next function or with termination.

If WAIT is not specified, ADADBS proceeds immediately after initiating the online process.

Example

Initiate an online process that reorders the Associator and Data Storage of file 10 first, and then file 11. The Associator padding factor of file 10 is to be 5 percent; the Data Storage padding factor of file 11 is to be 10 percent.

```
ADADBS ONLREORFILE FILE=10,ASSOPFAC=5
ADADBS             FILE=11,DATAPFAC=10
```

OPERCOM : Adabas Operator Commands

The OPERCOM function issues operator commands to the Adabas nucleus.

In an Adabas cluster environment, OPERCOM commands can be directed to a single cluster nucleus or to all active nuclei in the cluster. If a particular nucleus is not specified, the command defaults to the local nucleus.

Adabas issues a message to the operator, confirming command execution.

ADADBS OPERCOM operator-command
[**NOUSERABEND**]
[**NUCID=**{ nuc-id | **0** }]
[**TEST**]

In this section, the discussion of the individual operator commands follows the discussion of the optional parameters, since some of the operator commands behave differently when issued in an Adabas cluster environment.

Using OPERCOM Commands in Cluster Environments

Some ADARUN parameters are “global”; that is, they must have the same values for all nuclei in a cluster. Of these, some are set at session initialization and cannot be changed. Others can be modified on a running system. OPERCOM commands that change these modifiable global parameter values are handled in a special way in cluster environments.

If an Adabas cluster nucleus changes one or more “global” parameters, that nucleus acquires a “parameter change lock”, makes the changes in its local parameter area, informs the other cluster nuclei of the changes and waits for a reply. The other cluster nuclei make the changes in their own local parameter areas and send an “acknowledge” message.

Five OPERCOM commands use the GLOBAL option to operate across all active nuclei in a cluster: ADAEND, CANCEL, FEOFCL, FEOFPL, and HALT. For example:

```
ADADBS OPERCOM ADAEND, GLOBAL
```

All other OPERCOM commands use the NUCID=0 option to operate across all active nuclei in the cluster.

Optional Parameters

GLOBAL : Operate Across All Active Cluster Nuclei

Five OPERCOM commands use the GLOBAL option to operate across all active nuclei in a cluster: ADAEND, CANCEL, FEOFCL, FEOFPL, and HALT. For example:

```
ADADBS OPERCOM ADAEND, GLOBAL
```

All other OPERCOM commands use the NUCID=0 option to operate across all active nuclei in a cluster.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NUCID : Cluster Nucleus ID

Any nucleus running in an Adabas nucleus cluster is allowed to run Adabas utilities such as ADADBS.

With certain exceptions, the NUCID parameter allows you to direct the ADADBS OPERCOM commands to a particular nucleus in the cluster for execution, just as though the command had been issued by a locally run ADADBS OPERCOM operation. You can route most OPERCOM commands to all nuclei in a cluster by specifying NUCID=0.

If NUCID is not specified in a cluster environment, the command is routed to the local nucleus.

TEST : Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; nor the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Operator Commands

ADAEND

ADAEND [**,GLOBAL**]

This command terminates an Adabas session normally. No new users are accepted after this command has been issued. ET logic updating is continued until the end of the current logical transaction for each user. After all activity has been completed as described above, the Adabas session is terminated.

In nucleus cluster environments, the GLOBAL option can be used to terminate the Adabas session in all active cluster nuclei.

ALOCKF

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

ALOCKF=file-number

Lock a file in advance to ensure that an EXU, EXF, or UTI user will have exclusive control of the specified file. The advance-lock prevents new transactions from using the file. Once all current users have stopped using the file, the exclusive-control user has the lock. Until then, the exclusive-control user must wait.

To remove the advance lock without running the utility, see the RALOCKF command.

This command is not available

- in single user mode; or
- for a read-only nucleus.

CANCEL

CANCEL [**,GLOBAL**]

Cancel the Adabas session immediately. All command processing is immediately suspended. A pending autorestart is in effect which in turn causes the autorestart routine to be executed during the initialization of the next Adabas session.

In nucleus cluster environments, the GLOBAL option can be used to cancel the Adabas session in all active cluster nuclei.

CLOGMRG

CLOGMRG={ **YES**|**NO** }

Switches automatic command log merging (ADARUN CLOGMRG parameter value) on or off in nucleus cluster environments.

The CLOGMRG command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

CLUFREEUSER

CLUFREEUSER [**,TNA=**max-time][**,UID=**userid][**,FORCE**][**,GLOBAL**][**,NUCID**=nucid]

Note:
The CLUFREEUSER command is only valid in cluster environments. It can be issued against the local nucleus only or, with the GLOBAL option, against all active and inactive nuclei in the cluster.

Delete leftover user table elements (UTEs) in common storage that are no longer associated with user queue elements (UQEs) in a nucleus where

TNA is a decimal number specifying the timeout value in seconds.

UTEs that are not used during the time specified may be deleted if other conditions are fulfilled.

If TNA is not specified, UTEs may be deleted without regard to their recent use.

UID is a character string or hexadecimal byte string as follows:

cccccccc	—where the argument is 1–8 letters, digits, or embedded '–' signs without surrounding apostrophes.
'cccccccc'	—where the argument is 1–8 characters with surrounding apostrophes.
X'xxxxxxxxxxxxxxxx'	—where the argument is an even number of 2–16 hexadecimal digits enclosed by X' '.

A character string must be enclosed in apostrophes if it contains characters other than letter, digits, or embedded '–' signs. If a specified character string is less than 8 characters long, it is implicitly padded with blanks. If a specified hexadecimal string is shorter than 16 hexadecimal digits, it is implicitly padded with binary zeros.

If the last 8 bytes of a user's 28-byte communication ID match a specific user ID or user ID prefix, that user's UTE may be deleted if other conditions are fulfilled.

If UID not specified, UTEs may be deleted regardless of their user IDs.

FORCE Leftover UTEs are to be deleted even if the users are due a response code 9, subcode 20.

If FORCE is not specified, such UTEs are not deleted.

Before using the FORCE parameter, ensure that the users owning the UTEs to be deleted will not expect any of their transactions to remain open.

GLOBAL	Leftover UTEs throughout the Adabas cluster are to be deleted if they are no longer associated with UQEs and are eligible according to the other specified parameters. Additionally and subject to the other rules, leftover UTEs are deleted if their assigned nuclei have terminated since their last use. If GLOBAL is not specified, only UTEs assigned to the local nucleus and used since the nucleus start are eligible for deletion.
NUCID	is used to indicate that the command is to be processed by a specific nucleus in the cluster.

CT

CT= timeout-limit

Dynamically override the ADARUN CT parameter value; that is, the maximum number of seconds that can elapse from the time an Adabas command has been completed until the results are returned to the user through interregion communication (which depends on the particular operating system being used). The minimum setting is 1; the maximum is 16777215.

In nucleus cluster environments, the CT command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

DAUQ

Display the user queue element (UQE) of each user who has executed at least one Adabas command within the last 15 minutes.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DCQ

Display all posted command queue elements (CQEs). Each CQE's user ID, job name, and buffer length is displayed.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DDIB

Display data integrity block (DIB). This block contains entries indicating which Adabas utilities are active and the resources being used by each utility. The DDIB function can be performed with either an active or an inactive nucleus.

In nucleus cluster environments, the information displayed by the DDIB command is global; the command can be run on any nucleus.

DDSF

Display Adabas Delta Save Facility (DSF) status. The Adabas nucleus displays the DSF status on the operator console as well as in the ADADBS job protocol.

This function is only available if the nucleus is run with the parameter ADARUN DSF=YES.

In nucleus cluster environments, the information displayed by the DDSF command is global; the command can be run on any nucleus.

DFILES

DFILES={ n | n1–nx | n1,n2,n3,n4,n5 }

Displays the number of access, update, EXU, and UTI users for the specified files. User types are totaled for each file, and are listed by file.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DFILUSE

DFILUSE= file-number

Displays the count of commands processed for the specified file so far during the current session.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DHQ

Display up to five hold queue elements.

DHQA

Display all hold queue elements (HQEs).

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DLOCKF

Display locked files.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DNC

Display the number of posted command queue elements (CQEs).

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DNH

Display the number of ISNs currently in the hold queue.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DNU

Display the number of current users.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DONLSTAT

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

Display status of each active reorder or invert online process together with the process ID.

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DPARM

Display the Adabas session parameters currently in effect.

DRES

Display the allocated pool space and the highest use level ('high water mark') reached so far during the current session by record count and by percent for the following resources:

- attached buffers (AB) – current allocation not supported
- command queue (CQ)
- format pool (FP)
- hold queue (HQ)
- pool for the table of ISNs (TBI)
- pool for the table of sequential commands (TBQ or TBLES)
- user queue (UQ)
- unique descriptor pool (DUQPOOL)
- security pool
- user queue file list pool
- work pool (WP)
- pool for global transaction IDs (XIDs; nonzero only with Adabas Transaction Manager)
- cluster block update "redo" pool (nonzero only for a cluster nucleus with ADARUN LRDP greater than zero)

The actual values are displayed in nucleus message ADAN28 described in the *Adabas Messages and Codes* manual.

DSTAT

Display the current Adabas nucleus operating status.

DTH

Display thread status.

DUQ

Display up to five active and inactive user queue elements.

DUQA

Display all user queue elements (UQEs).

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

DUQE

DUQE=X'user-id**'**

Display a user queue element for the specified Adabas-assigned user ID as follows:

DUQE=X'A3CF2'

The user ID must be entered in hexadecimal format. Do not use a job name for the user ID.

In nucleus cluster environments, NUCID must always be specified because the user ID is not unique to the cluster.

DUUQE

Display utility user queue elements (UQEs).

In nucleus cluster environments, the NUCID=0 option can be used to display information for all active cluster nuclei. Information is displayed for each nucleus, one after the other.

FEOFCL

FEOFCL [**,GLOBAL**]

Close the current dual or multiple command log and switch to the other dual or another multiple command log. This command is valid only if dual or multiple command logging is in effect.

In nucleus cluster environments, the GLOBAL option can be used to switch the dual or multiple command log in all cluster nuclei at the same time.

FEOFPL

FEOFPL [**,GLOBAL**]

Close the current dual or multiple data protection log and switch to the other dual or another multiple protection log. This command is valid only if dual or multiple data protection logging is in effect.

In nucleus cluster environments, the GLOBAL option can be used to switch the dual or multiple protection log in all cluster nuclei at the same time.

HALT

HALT [**,GLOBAL**]

Stop Adabas session. A BT (backout transaction) command is issued for each active ET logic user. The Adabas session is then terminated; no dumps are produced.

In nucleus cluster environments, the GLOBAL option can be used to halt the Adabas session in all active cluster nuclei.

LOCKF

LOCKF= file-number

Lock the specified file. The specified file will be locked at all security levels.

LOCKU

LOCKU= file-number

Lock the specified file for all non-utility use. Adabas utilities can use the file normally.

LOCKX

LOCKX= file-number

Lock the specified file for all users except EXU or EXF users. EXU and EXF users can use the file normally. The lock is released automatically when an EXU user issues an OP command.

LOGGING

Start command logging.

LOGxx

Begin logging as indicated by “xx” for each command logged where “xx” is one of the following:

CB	the Adabas control block
FB	the Adabas format buffer
IB	the Adabas ISN buffer
IO	Adabas I/O activity
RB	the Adabas record buffer
SB	the Adabas search buffer
UX	user data passed in the seventh parameter of the Adabas parameter list
VB	the Adabas value buffer

NOLOGGING

Stop or prevent command logging.

NOLOGxx

Stop or prevent logging of “xx” where “xx” is one of the following:

CB	the Adabas control block
FB	the Adabas format buffer
IB	the Adabas ISN buffer

IO	Adabas I/O activity
RB	the Adabas record buffer
SB	the Adabas search buffer
UX	user data passed in the seventh parameter of the Adabas parameter list
VB	the Adabas value buffer

ONLRESUME

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

ONLRESUME=X'identifier**'**

Resume a previously suspended online reorder or invert process.

In a cluster environment, NUCID must always be specified because the online process ID is not unique to the cluster.

ONLSTOP

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

ONLSTOP=X'identifier**'**

Stop an online reorder or invert process cleanly. The process continues up to its next interrupt point in order to produce a consistent state, and then terminates after performing all necessary cleanup.

In a cluster environment, NUCID must always be specified because the online process ID is not unique to the cluster.

ONLSUSPEND

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

ONLSUSPEND=X'identifier**'**

Suspend an online reorder or invert process. The process continues up to its next interrupt point in order to produce a consistent state, performs a command throwback, and enters a state where it cannot be selected for processing. This command is useful if the online process is consuming too much of the nucleus resources.

In a cluster environment, NUCID must always be specified because the online process ID is not unique to the cluster.

RALOCKF

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

RALOCKF=n

Remove the advance lock on the specified file (see ALOCKF command) without running the utility.

RALOCKFA

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

Remove the advance lock on all files for which it has been set (see ALOCKF command) without running the utility.

RDUMPST

Terminate online dump status. This command is normally used if online execution of the ADASAV utility has terminated abnormally.

READONLY

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

READONLY={ **YES**|**NO** }

Switches READONLY status on or off.

In nucleus cluster environments, the READONLY command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

REVIEW

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

REVIEW={**NO** | **LOCAL** | hub-id}

Deactivate Adabas Review; change from hub mode to local mode; specify or change the Adabas Review hub with which a nucleus communicates.

STOPF

STOPF= file-number[, **PURGE**]

Stop users who are using the specified file. Only one file number can be specified. This command does not stop EXF or UTI users.

The optional PURGE parameter removes stopped user queue elements from the user queue when ADARUN OPENRQ=NO was specified. The following is an example of using the PURGE parameter:

ADADBS OPERCOM STOPF=5,PURGE

STOPI

STOPI= time [**,PURGE**]

Stop users who have not executed a command during the past “time” (in seconds). This command does not stop EXF or UTI users.

The optional PURGE parameter removes stopped user queue elements from the user queue when ADARUN OPENRQ=NO was specified. The following is an example of using the PURGE parameter:

ADADBS OPERCOM STOPI=3600,PURGE

STOPU

STOPU={ **X’**user-id**’**| job-name }

Note:
The STOPU=X’userid’ command is not allowed for online ADAORD or ADAINV processes. See the ONLSTOP=X’identifier ’ command instead.

Stop the user with the Adabas-assigned user ID (in the form shown in the display commands), or stop all users with the job “job-name”.

STOPU clears inactive or timed-out users, and deletes the user’s user queue element (UQE). If the program/user is an ET logic user, is not in ET status, and has not been stopped before STOPU is issued, Adabas backs out all updates made by the transaction to this point and releases all held records. If the transaction continues, only those changes following the STOPU are completed.

The user ID must be specified in hexadecimal format; for example:

STOPU=X’A3CF2’

In a cluster environment, NUCID must always be specified because the user ID is not unique to the cluster.

SYNCC

Force resynchronization of all ET users on the nucleus. The nucleus waits for all ET users to reach ET status before continuing.

TNAu

TNAu**=** time

Set non-activity time limit (in seconds) for users where “u” is one of the following:

A	for access-only (ACC) users
E	for ET logic users
X	for exclusive control (EXF/EXU) users

If specified, “time” must be a value greater than zero; it overrides the ADARUN value.

In nucleus cluster environments, the TNAu commands are global by definition and affect all nuclei in the cluster. If a NUCID is specified, it is ignored.

TT

TT= time

Set transaction time limit (in seconds) for ET logic users. If specified, this value must be greater than zero; it overrides the ADARUN value.

In nucleus cluster environments, the TT command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

UNLOCKF

UNLOCKF= file-number

Unlock the specified file and restore its usage to the prelocked status.

UNLOCKU

UNLOCKU= file-number

Unlock the specified file for utility use and restore it to its prelocked status for non-utility users.

UNLOCKX

UNLOCKX= file-number

Unlock the specified file and restore its usage to the prelocked status.

UTIONLY

Note:
Not currently available for use with Adabas Parallel Services cluster nuclei.

UTIONLY={ **YES**|**NO** }

Switch UTIONLY status on or off.

In nucleus cluster environments, the UTIONLY command is global by definition and affects all nuclei in the cluster. If a NUCID is specified, it is ignored.

PRIORITY : Change User Priority

The PRIORITY function may be used to set or change the Adabas priority of a user. A user's priority can range from 0 (the lowest priority) to 255 (the highest priority).

The user is identified by the same user ID provided in the Adabas control block (OP command, additions 1 field).

ADADBS PRIORITY **USERID=**'user-id'
[**NOUSERABEND**]
[**PRTY=**{ n|**255** }]
[**TEST**]

Essential Parameter

USERID : User ID

The user ID in the checkpoint file of the user for which priority is to be changed. If a record for this user does not exist, a new one is added to the checkpoint file.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PRTY : User Priority

The priority to be in effect for the user. A value in the range 0 for lowest priority to 255 for the highest priority may be specified. The default is 255. This value will be added to the operating system priority by the interregion communications mechanism.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

```
ADADBS PRIORITY USERID='USER24',PRTY=7
```

Set the priority assignment for the user with the user ID "USER24" to 7.

RECOVER : Recover Space

The RECOVER function recovers space allocated by rebuilding the free space table (FST). RECOVER subtracts file, DSST, and alternate RABN extents from the total available space.

```
ADADBS RECOVER      [NOUSERABEND]
                    [TEST]
```

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

REFRESH : Set File to Empty Status

The REFRESH function sets the file to 0 records loaded; sets the first extent for the address converter, Data Storage, normal index, and upper index to “empty” status; and deallocates other extents.

When the REFRESH function completes successfully, any locks previously set with the operator commands LOCKU or LOCKF are reset.

ADADBS REFRESH **FILE=**file-number
[**NOUSERABEND**]
[**PASSWORD=**'password']
[**TEST**]

Essential Parameter

FILE : File Number

FILE specifies the file that is to be set to “empty” status.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

This parameter is required if the file is password-protected.

TEST : Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

File 116 is to be set to empty status.

```
ADADBS REFRESH FILE=116
```

REFRESHSTATS : Refresh Statistical Values

The REFRESHSTATS function resets statistical values maintained by the Adabas nucleus for its current session. Parameters may be used to restrict the function to particular groups of statistical values.

When you invoke REFRESHSTATS, Adabas automatically writes the nucleus shutdown statistics to DD/PRINT.

Important:
Refreshing Adabas statistical values affects the corresponding Adabas Statistics Facility (ASF) field values. These values, which normally reflect the period from the start of the nucleus, will then refer to the time after the last refresh. ASF users may therefore find it useful to store the nucleus records with the appropriate ASF function before refreshing the values.

```
ADADBS  REFRESHSTATS   [ALL]
                       [CMDUSAGE]
                       [COUNTERS]
                       [FILEUSAGE]
                       [ NUCID=nucid ]
                       [NOUSERABEND]
                       [POOLUSAGE]
                       [THREADUSAGE]
```

Optional Parameters

ALL : All Statistical Values

The ALL keyword may be specified as an abbreviation for the combination of CMDUSAGE, COUNTERS, FILEUSAGE, POOLUSAGE, and THREADUSAGE.

If none of the option keywords is specified, ALL is the default option.

CMDUSAGE : Command Usage Counters

The CMDUSAGE parameter is specified to reset the counters for Adabas direct call commands such as Lx, Sx, or A1.

COUNTERS : Frequency Counters

The COUNTERS parameter is specified to reset the counter fields for local or remote calls, format translations, format overwrites, Autorestarts, protection log switches, buffer flushes, and command throw-backs.

FILEUSAGE : Count of Commands Per File

The FILEUSAGE parameter is specified to reset the count of commands for each file.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NUCID : Cluster Nucleus ID

Any nucleus running in an Adabas nucleus cluster is allowed to run Adabas utilities such as ADADBS. The NUCID parameter allows you to direct the ADADBS REFRESHSTATS function to a particular nucleus in the cluster for execution, just as though the command had been issued by a locally run ADADBS REFRESHSTATS operation.

If you specify NUCID=0, the statistical values are refreshed for all active nuclei in the cluster.

POOLUSAGE : High-Water Marks for Nucleus Pools

The POOLUSAGE parameter is specified to reset the high-water marks for the nucleus pools such as the work pool, the command queue, or the user queue.

THREADUSAGE : Count of Commands Per Thread

The THREADUSAGE parameter is specified to reset the count of commands for each Adabas thread.

Example

File 116 is to be set to empty status.

```
ADADBS  REFRESHSTATS  CMDUSAGE,POOLUSAGE,NUCID=3
```

After the shutdown statistics for the Adabas cluster nucleus with NUCID=3 are written to DD/PRINT, the command counters and the pool high-water marks for the nucleus are reset.

RELEASE : Release Descriptor

The RELEASE function releases a descriptor from descriptor status.

This function results in the release of all space currently occupied in the Associator inverted list for this descriptor. This space can then be reused for this file by reordering or ADALOD UPDATE. No changes are made to Data Storage.

When releasing descriptor space for an Adabas expanded file, perform the RELEASE function for **each individual component file** of the expanded file. Each RELEASE operation on a component file causes a message that confirms the change, and returns condition code 4.

```
ADADBS   RELEASE      FILE=file-number
                      DESCRIPTOR='name'
                      [NOUSERABEND]
                      [PASSWORD='password']
                      [TEST]
```

Essential Parameters

FILE : File Number

FILE specifies the file that contains the descriptor to be released. The file cannot be an Adabas system file.

DESCRIPTOR : Descriptor to Be Released

DESCRIPTOR specifies the descriptor to be released. Any descriptor type can be specified. A descriptor currently being used as the basis for file coupling cannot be specified. If the descriptor being released is an ADAM descriptor, the file is no longer processed as an ADAM file.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

This parameter is required if the file is password-protected. Specify the password between apostrophes (’).

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

```
ADADBS RELEASE FILE=31,DESCRIPTOR='AA'
```

Descriptor AA in file 31 is released from descriptor status.

RENAME : Rename File/Database

The RENAME function may be used to change the name assigned to a file or database.

```
ADADBS  RENAME      NAME='name'
                    [FILE=file-number]
                    [NOUSERABEND]
                    [PASSWORD='password']
                    [TEST]
```

Essential Parameter

NAME : New File Name

NAME is the new name to be assigned to the file. It is specified between apostrophes (for example, 'RESERVATIONS'). A maximum of 16 characters can be used.

Optional Parameters

FILE : File Number

FILE is the number of the file to be renamed: if specified as zero or omitted, the database is renamed.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

The password of the file. This parameter is required if the file is password-protected.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Examples

The name of file 2 is to be changed to “INVENTORY”.

```
ADADBS RENAME FILE=2,NAME='INVENTORY'
```

The database is renamed to “RESERVATIONS”.

```
ADADBS RENAME NAME='RESERVATIONS',FILE=0
```

RENUMBER : Change File Number

The RENUMBER function changes the number of an Adabas file.

```
ADADBS  RENUMBER   FILES=current-number, new-number
                   [NOUSERABEND]
                   [TEST]
```

Essential Parameter

FILES : Current File Number, New File Number

The number currently assigned to the file, and the new number to be assigned to the file. If the new number is assigned to another file, the RENUMBER function will not be performed.

An Adabas system file cannot be used. The file may not be security-protected, may not be coupled to another file, and may not be part of an expanded file.

Optional Parameter

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

The file number for file 4 is to be changed to 40.

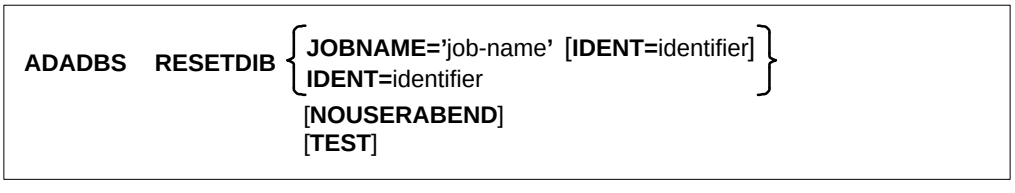
```
ADADBS RENUMBER FILES=4,40
```

RESETDIB : Reset Entries in Active Utility List

The RESETDIB function resets entries in the active utility list (that is, the data integrity block or DIB).

Adabas maintains a list of the files used by each Adabas utility in the DIB. The DDIB operator command (or Adabas Online System) may be used to display this block to determine which jobs are using which files. A utility removes its entry from the DIB when it terminates normally. If a utility terminates abnormally (for example, the job is cancelled by the operator), the files used by that utility remain "in use". The DBA may release any such files with the RESETDIB function.

Note:
The RESETDIB function can be executed either with or without an active nucleus. To remove a DIB from an abended ADAORD REORDB, REORDATA, REORASSO, ADADBS RESETDIB has to run without an active nucleus.



Essential Parameters

JOBNAME : Job Name

This parameter specifies the name of the job whose entry is to be reset. If it is not unique, the IDENT parameter must also be specified.

IDENT : Utility Execution Identifier

A unique number that identifies a utility execution. It may be specified alone or to qualify a job name when the same name has been used for various utility executions. The identifier may be obtained using the operator command DDIB or Adabas Online System.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Examples

The entry in the DIB block for job “JOB1” is to be deleted.

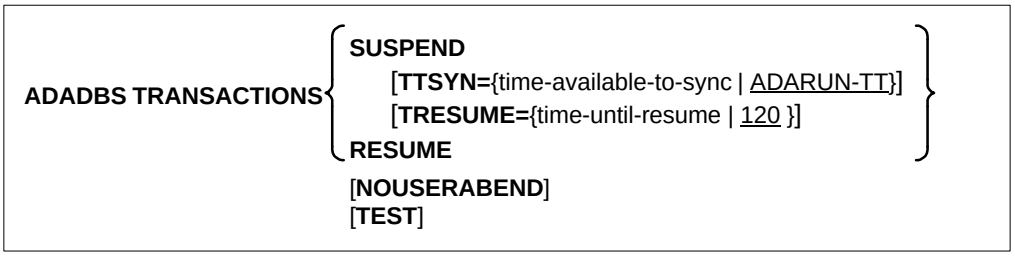
```
ADADBS RESETDIB JOBNAME='JOB1'
```

The entry in the DIB block for “JOB2” with IDENT=127 is to be deleted.

```
ADADBS RESETDIB JOBNAME='JOB2',IDENT=127
```

TRANSACTIONS : Suspend and Resume Transactions

The TRANSACTIONS function may be used to suspend and resume update transaction processing; that is, to reach a quiesced state that could be a recoverable starting point.



Once the SUSPEND function has been submitted, new update transactions are not held in the user queue but in the command queue. Executing transactions are allowed to finish if they can do so within the time allotted by the TTSYN parameter. Any transactions that exceed this time are backed out. In a cluster environment, all cluster nuclei are likewise quiesced.

Once the quiesce is successful, the buffers are flushed for all nuclei so that the DASD files are current with the content of the buffers. A checkpoint SYNC-73 is written and ADADBS is notified.

At this point, the user may execute a non-Software AG fast backup product such as IBM's FlashCopy or StorageTek's SnapShot to "COPY" off the database; that is, copy pointers to the data created by the fast backup product in the electronic memory of the array storage device.

Warning:
Software AG does not recommend using such a database fastcopy as a substitute for a regular Software AG database (or delta) save. Not only does Software AG have no control over the datasets that are included in the database fastcopy, but it also cannot vouch for the success of the fastcopy. Moreover, delta saves cannot sensibly be run on a copy of the database, as the DSF status change effected by the delta save would occur on the database copy instead of the original.

If the COPY completes before the TRESUME timeout and the RESUME function is issued, the nucleus writes a SYNS-74 checkpoint, leaves the suspended state and resumes update processing. The database was in a valid state over the whole duration of the COPY process.

If the COPY does not complete before the TRESUME timeout, Adabas automatically leaves the suspended state and resumes update processing. If the RESUME function is issued subsequently, Adabas rejects it with a response code and ADADBS terminates abnormally with an error message. This means that whatever COPY has been produced while update processing was suspended is invalid and must not be used, because Adabas may have resumed updating the database while the COPY process was still in progress.

If the so-created copy of the database is used for recovery, removing the need to restore the database as of the time of the COPY, the subsequent regenerate should be started at the SYNC-73 checkpoint written at the end of the SUSPEND function.

Important:
In a job where a SUSPEND function is followed by other job steps and then by a RESUME function, none of the job steps in between should be update-type commands or functions; otherwise, job execution will stall until the nucleus times out the suspended state.

Essential Parameters

SUSPEND : Suspend Transactions and Quiesce the Database

Use this parameter to suspend update transaction processing and quiesce the database.

RESUME : Resume Transaction Processing that was Previously Suspended

Use this parameter to resume update transaction processing that was previously suspended. If this parameter is used while Adabas is not in a suspended state or is no longer in a suspended state, this function terminates with an error.

Optional Parameters

TRESUME

Use this parameter to specify the amount of time in seconds the system is to remain quiesced after being suspended before the nucleus automatically resumes normal update transaction processing. If this parameter is not specified, the default is 120 seconds and the maximum is 86400 seconds or 24 hours. The count begins when the nucleus has been successfully quiesced.

TTSYN

Use this parameter to specify the maximum amount of time the nucleus is to wait for all ET users to reach ET status before it forcibly ends and backs out update transactions that are still running in order to quiesce the system. If this parameter is not specified, the default is the ADARUN TT value.

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

Quiesce a database allowing 300 seconds for the currently running update transactions to finish and 150 seconds thereafter for the suspension to last before Adabas automatically resumes normal processing:

```
ADADBS TRANSACTIONS SUSPEND TTSYN=300,TRESUME=150
```

UNCOUPLE : Uncouple Files

The UNCOUPLE function is used to eliminate the coupling relationship between two files.

```
ADADBS  UNCOUPLE   FILES=number, number
                   [NOUSERABEND]
                   [PASSWORD='password']
                   [TEST]
```

Essential Parameter

FILES : Files to Be Uncoupled

FILES specifies the two files to be uncoupled.

Optional Parameters

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

PASSWORD specifies the security password for one or both files, and is required if either of the files is password-protected. If both files are password-protected, the password applies to both files. The password must be enclosed in single quotation marks.

TEST : Test Syntax

This parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables. See page 138 for more information on using the TEST parameter in ADADBS functions.

Example

Files 62 and 201 are to be uncoupled. One or both are protected with the password “PAIR05”.

```
ADADBS UNCOUPLE FILES=62,201,PASSWORD='PAIR05'
```

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADADBS with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems, and shows examples of each of the job streams.

Collation with User Exit

If a collation user exit is to be used during ADADBS ONLINVERT execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding support (UES), the format of the collation descriptor user exit parameter is

```
ADARUN CDXnn=exit-name
```

—where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01–08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *DBA Reference Manual* for more information.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn		Required for OPERCOM DDIB or RESETDIB with inactive nucleus
ADARUN parameters	SYSDTA/ DDCARD		Operations Manual
ADADBS parameters	SYSDTA/ DDKARTE		Utilities Manual
ADARUN messages	SYSOUT/ DDPRINT		Messages and Codes
ADADBS messages	SYSLST/ DDDRUCK		Messages and Codes

ADADBS JCL Example (BS2000)

In SDF Format:

```
/.ADADBS LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A D B S ALL FUNCTIONS
/REMARK *
/ASS-SYSLST L.DBS.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADADBS,DB=yyyyy,IDTNAME=ADABAS5B
ADADBS REFRESH FILE=1
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADADBS LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A D B S ALL FUNCTIONS
/REMARK *
/SYSFILE SYSLST=L.DBS
/FILE ADA.MOD,LINK=DDLIB
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADADBS,DB=yyyyy,IDTNAME=ADABAS5B
ADADBS REFRESH FILE=1
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	Required only for OPERCOM DDIB or RESETDIB functions with inactive nucleus
ADADBS messages	DDDRUCK	printer	Messages and Codes
ADARUN messages	DDPRINT	printer	Messages and Codes
ADARUN parameters	DDCARD	reader	Operations Manual
ADADBS parameters	DDKARTE	reader	

ADADBS JCL Example (OS/390 or z/OS)

Refer to ADADBS in the MVSJOBS dataset for this example.

```
//ADADBS    JOB
//*
//*     ADADBS:
//*        DATA BASE SERVICES (BATCH)
//*
//DBS       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD  <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADADBS,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADADBS      REFRESH FILE=1
/*
//
```

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk/terminal/reader	Required only for OPERCOM DDIB or RESETDIB functions with inactive nucleus
ADARUN parameters	DDCARD	disk/terminal/reader	Operations Manual
ADADBS parameters	DDKARTE	disk/terminal/reader	
ADARUN messages	DDPRINT	disk/terminal/printer	Messages and Codes
ADADBS messages	DDDRUCK	disk/terminal/printer	

ADADBS JCL Example (VM/ESA or z/VM)

Refer to ADADBS in the MVSJOBS dataset for this example.

```
DATADEF DDPRINT,DSN=ADADBS,DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADADBS.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNDBS.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADADBS.CONTROL,MODE=A
ADARUN
```

Contents of RUNDBS CONTROL A1:

```
ADARUN PROG=ADADBS,DEVICE=dddd,DB=yyyyy
```

Contents of ADADBS CONTROL A1:

```
ADADBS REFRESH FILE=1
```

VSE/ESA

File	File Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	Required for OPERCOM DDIB or RESETDIB functions with inactive nucleus
ADARUN parameters	– CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADADBS parameters	–	reader	SYSIPT	Utilities Manual
ADARUN messages	–	printer	SYSLST	Messages and Codes
ADADBS messages	–	printer	SYS009	Messages and Codes

***** Any programmer logical unit may be used.

ADADBS JCS Example (VSE/ESA)

See appendix B for a description of the VSE/ESA procedures.

Refer to member ADADBS.X in the MVSJOBS dataset for this example.

```
* $$ JOB JNM=ADADBS,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
*       DATABASE SERVICES (BATCH)
// JOB ADADBS
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADBS,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADADBS REFRESH FILE=1
/*
/&
* $$ EOJ
```

6

ADADCK : CHECK DATA STORAGE

Functional Overview

ADADCK checks the Data Storage and the Data Storage space table (DSST) of a specific file (or files) in the database.

ADADCK reads each used Data Storage block (according to the Data Storage extents in the file control block) and performs the following checks:

- Block length within permitted range? ($4 \leq$ block length $\leq$ physical block size)
- Sum of length of all records in the Data Storage block plus 4 = block length?
- Is there any record with a record length greater than the maximum compressed record length for the file or with a length ≤ 0?
- Are there any duplicate ISNs within one block?
- Does the associated DSST element contain the correct value? If not, a REPAIR of the DSST is necessary (see REPAIR parameter on page 221).

Notes:

1. *ADADCK does not require the Adabas nucleus to be active.*
2. *If the nucleus is active, ADADCK synchronizes its operation with the active nucleus unless the NOOPEN parameter is specified.*
3. *Any pending autorestart condition is ignored.*
4. *This utility should be used only for diagnostic purposes.*

ADADCK returns a condition code 4 or 8 if an error occurs.

DSCHECK : Check Data Storage

ADADCK DSCHECK [**FILE=** { file [**FROMRABN=**DS-blknum] [**TORABN=**DS–blknum] | file–file }]
[**NOOPEN**]
[**NOUSERABEND**]
[**REPAIR**]
[**USAGE**]

Optional Parameters and Subparameters

FILE : Files to Be Checked

The file (or a single range of files) to be checked. If omitted, all files in the database are checked.

FROMRABN : Data Storage Block Number

The RABN of the Data Storage block where the check is to start. This parameter is applicable only if a single file is to be checked.

If omitted, the check starts at the beginning of the first allocated Data Storage extent for the file.

NOOPEN : Prevent Open Synchronization

When starting, ADADCK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADADCK from issuing the open call and blocking file usage for other users.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

REPAIR : Repair the Data Storage Space Table

If ADADCK finds any invalid Data Storage space table elements, it automatically repairs the table if this parameter is supplied.

TORABN : Ending Data Storage Block Number

The RABN of the Data Storage block where the check is to end. This parameter is applicable only if a single file is to be checked.

USAGE : Print Data Storage Block Usage

If USAGE is specified, ADADCK prints a bar graph that shows the number of bytes used in each Data Storage block, the block size, and the percentage of blocks used.

Examples

Check Data Storage and its space table for file 20, print a bar graph of the Data Storage block utilization and repair the space table if required.

```
ADADCK DSCHECK FILE=20, USAGE, REPAIR
```

Check Data Storage and its space table for the files 8 through 12.

```
ADADCK DSCHECK FILE=8–12
```

Check Data Storage and its space table for file 12 in the RABN range 878 through 912.

```
ADADCK DSCHECK FILE=12, FROMRABN=878,TORABN=912
```

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADADCK with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems and shows examples of each of the job streams.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations Manual*
ADADCK parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADADCK messages	SYSLST DDDRUCK		*Messages and Codes*

ADADCK JCL Example (BS2000)

In SDF Format:

```
/.ADADCK LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK *A D A D C K DATA STORAGE CHECK
/REMARK *
/REMARK *
/ASS-SYSLST L.DCK.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADADCK,DB=yyyyy,IDTNAME=ADABAS5B
ADADCK DSCHECK FILE=27
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADADCK LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK *A D A D C K DATA STORAGE CHECK
/REMARK *
/REMARK *
/SYSFILE SYSLST=L.DCK.DATA
/FILE ADA.MOD,LINK=DDLIB


/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADADCK,DB=yyyyy,IDTNAME=ADABAS5B
ADADCK DSCHECK FILE=27
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADADCK parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADADCK messages	DDDRUCK	printer	*Messages and Codes*

ADADCK JCL Example (OS/390 or z/OS)

Refer to ADADCK in the MVSJOBS dataset for this example.

```
//ADADCK    JOB
//*
//*    ADADCK:
//*          DATA STORAGE CHECK
//*
//DCK       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADADCK,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADADCK DSCHECK FILE=27
/*
//
```

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	DDCARD	disk/terminal/reader	*Operations Manual*
ADADCK parameters	DDKARTE	disk/terminal/reader	
ADARUN messages	DDPRINT	disk/terminal/printer	*Messages and Codes*
ADADCK messages	DDDRUCK	disk/terminal/printer	*Messages and Codes*

ADADCK JCL Example (VM/ESA or z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPRINT,DSN=ADADCK.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADADCK.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNDCK.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADADCK.CONTROL,MODE=A
ADARUN
```

Contents of RUNDCK CONTROL A1:

```
ADARUN PROG=ADADCK,DEVICE=dddd,DB=yyyyy
```

Contents of ADADCK CONTROL A1:

```
ADADCK DSCHECK FILE=27
```

VSE/ESA

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
ADARUN parameters	— CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADADCK parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	*Messages and Codes*
ADADCK messages		printer	SYS009	*Messages and Codes*

***** Any programmer logical unit may be used.

ADADCK JCS Example (VSE/ESA)

See appendix B for descriptions of the VSE/ESA procedures (PROCs).

Refer to member ADADCK.X for this example.

```
* $$ JOB JNM=ADADCK,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADADCK
*       DATA STORAGE CHECK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADCK,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADADCK DSCHECK FILE=27
/*
/&
* $$ EOJ
```

7

ADADEF : DEFINE A DATABASE

Functional Overview

The ADADEF utility is used to

Function	Action	Page
DEFINE	define a new database and checkpoint system file; set default encodings for the new database	228
MODIFY	change default encodings set using ADADEF DEFINE	237
NEWWORK	define a new Work file for an existing database	240

The following database characteristics are defined with ADADEF:

- database name and ID
- database components (Associator, Data Storage, and Work)
  - device type
  - size
- checkpoint system file
- database default encodings

Database Components

Each database component (Associator, Data Storage, and Work) must be formatted by the ADAFRM utility before it is defined with ADADEF. The ADADEF utility may also be used to define a new Work dataset for an existing database.

Systems using the Recovery Aid feature require a recovery log (RLOG) dataset, which must first be formatted with the ADAFRM utility, and then defined using the ADARAI utility.

Checkpoint File

Adabas uses the checkpoint system file to store checkpoint data and user data provided with the Adabas CL and ET commands. It is required and must be specified using the ADADEF DEFINE (database) function.

DEFINE : Defining a Database and Checkpoint File

Syntax

The database and the checkpoint file must be defined at the same time.

The database parameters include the required ASSOSIZE, DATASIZE, and WORKSIZE parameters and the optional (non-indented) parameters ASSODEV through WORKDEV shown in the syntax diagram.

The FILE=...,CHECKPOINT,... statement is also required for database definition. The checkpoint file parameters (indented under the FILE statement in the syntax diagram) should be specified immediately following the FILE statement. See the examples on page 236.

ADADEF DEFINE **ASSOSIZE=**size-list
DATASIZE=size-list
WORKSIZE=size
FILE=file-number**,CHECKPOINT**
DSSIZE=size
MAXISN=maximum-number-of-records-expected
[**ACRABN=**starting-rabn]
[**ASSOPFAC=**{ Associator-padding-factor | **10** }]
[**ASSOVOLUME=** 'Associator-extent-volume']
[**DATAPFAC=**{ Data-Storage-padding-factor | **10** }]
[**DATAVOLUME=** 'Data-Storage-extent-volume']
[**DSDEV=**device-type]
[**DSRABN=**starting-rabn]
[**DSREUSE=**{ **NO** | **YES** }]
[**ISNSIZE=**{ **3** | **4** }]
[**MAXDS=**maximum-Data-Storage-secondary-allocation]
[**MAXNI=**maximum-normal-index-secondary-allocation]
[**MAXUI=**maximum-upper-index-secondary-allocation]
[**NAME=**{ 'file-name' | **CHECKPOINT** }]
[**NIRABN=**starting-rabn]
[**NISIZE=**size]

[**UIRABN=**starting-rabn]
[**UISIZE=**size]
[**ASSODEV=**{ device-type-list | ADARUN-device *}*]
[**DATADEV=**{ device-type-list | ADARUN-device *}*]
[**DBIDENT=**{ database-id | ADARUN-dbid *}*]
[**DBNAME=**{ database-name | **GENERAL–DATABASE** *}*]
[**FACODE=**{alpha-EBCDIC-key | **37**}]
[**FWCODE=**{wide-key | **4095**}]
[**MAXFILES=**{ maximum-number-of-files | **255** }]
[**NOUSERABEND**]
[**OVERWRITE**]
[**RABNSIZE=**{ **3**|**4** }]
[**UACODE=**{alpha-ASCII-key | **437**}]
[**UES=**{**YES** | **NO**}]
[**UWCODE=**{wide-key | FWCODE-definition}]
[**WORKDEV=**{ device-type-list | ADARUN -device *}*]

Essential Parameters

ASSOSIZE / DATASIZE / WORKSIZE : Database Size

ASSO-/DATA-/WORKSIZE specifies the number of blocks or cylinders to be assigned to the Associator, Data Storage, or Work. A block value must be followed by “B”; otherwise, the value is assumed to be cylinders.

If the Associator or Data Storage is to be contained on more than one dataset, the size of each dataset must be specified. If a companion ASSODEV or DATADEV parameter specifies two or more extents, the equivalent ASSOSIZE or DATASIZE parameter must specify the extent sizes as positional operands in the corresponding order (see the examples starting on page 236).

The minimum WORKSIZE allowed is 300 blocks.

Note:
If ASSOSIZE or DATASIZE is not specified, the ADADEF DEFINE function will not execute. If WORKSIZE is not specified, the function will allocate three (3) cylinders to the Work dataset. Because 3 cylinders are usually not enough to start the database, WORKSIZE is considered to be a required parameter.

DSSIZE : Data Storage Size

DSSIZE specifies the number of blocks or cylinders to be assigned to checkpoint/Data Storage. For blocks, the value specified must be followed by “B” (for example, DSSIZE=80B).

The size of the checkpoint file specified with the DSSIZE and MAXDS parameters depends on

- the amount of ET data to be stored;
- the number of utility runs for which checkpoint information is to be retained;
- the number of user IDs.

FILE . . . CHECKPOINT Parameter

FILE= file-number **, CHECKPOINT**

The FILE...CHECKPOINT parameter indicates the file number to be used for the checkpoint system file. This parameter is required; the file number must be 255 or lower.

Adabas uses the checkpoint system file to store checkpoint data and user data provided with the Adabas CL and ET commands.

MAXISN : Highest ISN to be Used

The highest ISN that may be assigned to the file. The value specified is used to determine the space allocation for the address converter. When determining the MAXISN, consider the importance of ET data and checkpoint data to your site.

Adabas considers ET data to be more important than checkpoint data. As soon as the ET data ISN range in the checkpoint system file is exhausted, the first checkpoint ISN is deleted and given to the ET data. This is an ongoing process. As soon as the MAXISN is reached, a new address converter extent is allocated and given to the checkpoint data. You can delete checkpoint data piece by piece using the Adabas Online System function DELCP.

Note:
The way the checkpoint handles data is subject to change in a future release of Adabas.

Optional Parameters

ACRABN / DSRABN / NIRABN / UIRABN : Starting RABN

These parameters may be used to cause allocation for their respective areas to begin with the specified RABN:

- ACRABN for the address converter
- DSRABN for Data Storage
- NIRABN for the normal index
- UIRABN for the upper index

ASSODEV / DATADEV / WORKDEV : Device Type

ASSO-/DATA-/WORKDEV specify the device type(s) to be assigned to the Associator, Data Storage, and Work. These parameters are required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter.

WORKDEV, if specified, can only be one device type. If the Associator (ASSODEV) or Data Storage (DATADEV) is to be contained on more than one dataset, the device type for each dataset must be specified, even if both extents are on the ADARUN DEVICE type.

If multiple extents are used with VSAM datasets, ASSODEV and DATADEV must reflect the dynamic device type; that is, DD/xxxxR1=9999; DD/xxxxR2=8888; ... DD/xxxxR5=5555. For example, when defining DDDATAR1 and DDDATAR2, DATADEV=9999,8888.

Space allocation for specified device types must be given in companion ASSOSIZE, DATASIZE, and WORKSIZE parameters on this or another ADADEF statement in the same job. If a ASSODEV or DATADEV parameter specifies more than one extent on the same or different device types *(*DATADEV=3380,3350, for example), the companion ASSOSIZE or DATASIZE parameter must specify the related extent sizes in corresponding order.

ASSOPFAC / DATAPFAC : Padding Factor

ASSOPFAC defines the percentage of space in each Associator RABN block to be reserved for later entries (padding space). This space is used for later descriptor extensions or ISN additions. The percentage value specified, which can range 1–90, should be large enough to avoid the overhead caused when block overflow forces splitting of an existing address block into two new blocks. If ASSOPFAC is not specified, ADADEF assumes a padding factor of 10%.

DATAPFAC defines the percentage of space in each Data Storage RABN block to reserve for later entries (padding space). This space is used when changes to an existing data record cause it to need more space in the block; an updated record that no longer fits in the existing block must be moved to another block. The percentage value specified, which can range 1-90, should be large enough to avoid the overhead caused when block overflow forces splitting of an existing address block into two new blocks. If DATAPFAC is not specified, ADADEF assumes a padding factor of 10%.

ASSOVOLUME / DATAVOLUME : Extent Volume

Note:
Values for ASSOVOLUME and DATAVOLUME must be enclosed in apostrophes.

ASSOVOLUME specifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) is to be allocated.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) are allocated.

If the requested number of blocks cannot be found on the specified volume, ADADEF retries the allocation while disregarding the ASSOVOLUME or DATAVOLUME parameter value.

If ACRABN, UIRABN, or NIRABN is specified, ADADEF ignores the ASSOVOLUME value when allocating the corresponding extent type.

If DSRABN is specified, DATAVOLUME is ignored for the related file.

If ASSOVOLUME and/or DATAVOLUME are not specified, the file's Associator and/or Data Storage space, respectively, is allocated according to ADADEF's default allocation rules.

DBIDENT : Database Identifier

DBIDENT specifies the identification number to be assigned to the database. A value in the range 1–65535 may be specified. If this parameter is omitted, the value specified with the ADARUN DBID parameter is used.

If multiple databases are to be established, the DBIDENT parameter is required in order to uniquely identify each database.

DBNAME : Database Name

DBNAME is the name to be assigned to the database. This name appears in the title of the Database Status Report produced by the ADAREP utility. A maximum of 16 characters may be specified. Enclose the name in single quotation marks if the name includes any special characters other than dashes, or if the name contains embedded blanks.

If this parameter is omitted, a default value of "GENERAL–DATABASE" is assigned.

DSDEV : Device Type for Data Storage

DSDEV specifies the device type to be used for the checkpoint file's Data Storage. There is no default value; if DSDEV is not specified, an arbitrary device type is used.

DSREUSE : Storage Reusage

DSREUSE indicates whether space which becomes available in the checkpoint file is to be reused. The default is YES.

FACODE : Encoding for Alphanumeric Fields

The FACODE parameter specifies the default encoding for alphanumeric fields for all files in the database. The encoding must be derived from EBCDIC encoding; that is, X'40' is the space character. Modal or "shift" type double-byte character set (DBCS) encodings are supported; fixed type DBCS (DBCS-only) encodings are not supported. The default encoding key is 37.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB.

You can change the default encoding set in this parameter using the ADADEF MODIFY function. Changing the database-wide setting does not affect files already loaded.

FWCODE : Encoding for Wide-Character Fields

The FWCODE parameter specifies the default encoding for wide-character (W) format fields for all files in the database. The default encoding is 4095; that is, Unicode.

The FWCODE parameter can be used to set a wide-character encoding that defines the superset of code points of all user encodings. For example, Unicode encompasses about 50,000 code points as opposed to Host-DBCS and Shift-JIS with about 10,000 code points each.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB.

You can change the default encoding set in this parameter using the ADADEF MODIFY function. Changing the database-wide setting does not affect files already loaded.

ISNSIZE : 3- or 4-Byte ISN

ISNSIZE indicates whether ISNs in the file are 3 or 4 bytes long. The default is 3 bytes.

MAXDS / MAXNI / MAXUI : Maximum Secondary Allocation

MAXDS/NI/UI specify the maximum number of blocks per secondary extent for Data Storage, the normal index, and the upper index, respectively. The value specified must be followed by "B" for blocks (for example, MAXDS=8000B) and cannot be more than 65535B.

MAXFILES : Highest File Number

MAXFILES specifies the maximum number of files that can be loaded into the database. The minimum value for this parameter is 3. The highest value permitted is 5000 or one less than the ASSOR1 blocksize, whichever is lower. For example, 2003 is the highest MAXFILES value for a database whose ASSOR1 is stored on a 3380 DASD.

The value specified determines the number of file control blocks and field definition tables to be allocated when the database is being established. Each file control block requires one Associator block and each field definition table requires four Associator blocks.

If this parameter is omitted, a value of 255 is assigned.

Once the database has been established, the value for MAXFILES may be changed only by executing the REORASSO or REORDB functions of the ADAORD utility.

NAME : Name of the Checkpoint File

NAME specifies the name for the checkpoint file being defined. This name appears on the Database Status Report produced by the ADAREP utility. The maximum number of characters permitted is 16. The default file name is CHECKPOINT.

NISIZE : Normal Index Size

NISIZE specifies the number of blocks or cylinders to be assigned to the normal index. For blocks, the value specified must be followed by "B" (for example, NISIZE=80B).

NOUSERABEND : Termination Without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

OVERWRITE : Overwrite Existing Database

Specify OVERWRITE to write over an existing database. OVERWRITE cannot be specified when creating a database with newly formatted datasets.

RABNSIZE : 3- or 4-Byte RABN

RABNSIZE specifies the length of RABNs in the database. Specify 3 for 24-bit RABNs or 4 for 31-bit RABNs. The default is 3.

UACODE : User Encoding for Alphanumeric Fields

The parameter UACODE specifies the default encoding for alphanumeric fields for ASCII users. The encoding must be derived from ASCII encoding; that is, X'20' is the space character. Encodings for multiple-byte character sets are supported. The default encoding is 437.

The UACODE value is not stored in the file being loaded.

You can override the default encoding set in this parameter for a user session using the OP command. You can change it generally using the ADADEF MODIFY function.

UES : Universal Encoding Support

Setting the parameter UES activates universal encoding support for the database. Any valid xxCODE parameter (FACODE, FWCODE, UACODE, UWCODE) implicitly sets UES=YES.

To deactivate UES, you must explicitly set UES=NO.

You can change the default setting of this parameter generally using the ADADEF MODIFY function.

UISIZE : Upper Index Size

UISIZE specifies the number of blocks or cylinders to be assigned to the upper index. For blocks, the value specified must be followed by “B” (for example, UISIZE=80B).

UWCODE : User Encoding for Wide-Character Fields

The UWCODE parameter specifies the user encoding for wide-character (W) format fields. If the parameter is not specified, the default value is the current value of FWCODE.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB.

You can override the default encoding set in this parameter for a user session using the OP command. You can change the default setting generally using the ADADEF MODIFY function. Changing the database-wide setting does not affect files already loaded.

Examples

Example 1:

```
ADADEF DEFINE
ADADEF  ASSOSIZE=200,DATASIZE=600,WORKSIZE=50
ADADEF  DBIDENT=1,DBNAME=DATABASE-1
ADADEF  MAXFILES=150
ADADEF  FILE=1,CHECKPOINT
ADADEF  NAME='DB1-CHECKPOINT',MAXISN=5000
ADADEF  DSSIZE=2,NISIZE=50B,UISIZE=10B
```

The Associator, Data Storage and Work sizes are equal to 200, 600 and 50 cylinders, respectively. The numeric identifier for the database is 1 and the database name is DATABASE-1. The maximum number of files (and the highest file number) that may be loaded into the database is 150. File 1 is to be reserved for the Adabas checkpoint file. The name of the first system file is to be DB1-CHECKPOINT. The Data Storage size for this file is to be 2 cylinders; the normal index size 50 blocks; the upper index size 10 blocks; and the MAXISN is to be 5000.

Example 2:

```
ADADEF  DEFINE
ADADEF  ASSODEV=3380,DATADEV=3380,3390,WORKDEV=3380
ADADEF  ASSOSIZE=100,DATASIZE=200,300,WORKSIZE=25
ADADEF  DBIDENT=2,DBNAME='DATABASE_2'
ADADEF  MAXFILES=255
ADADEF  FILE=255,CHECKPOINT,MAXISN=5000
ADADEF  DSSIZE=3,NISIZE=100B,UISIZE=20B
```

The Associator is to be contained on a 3380 device type, and occupies 100 cylinders. Data Storage comprises two datasets: the first dataset is 200 cylinders contained on the first DATADEV (3380) device type, and the second dataset is 300 cylinders contained on the second DATADEV (3390) device type. The Work space is 25 cylinders on the WORKDEV device (3380).

The numeric identifier for the database is 2, and the database name is DATABASE_2. A maximum of 255 files may be loaded into the database. An Adabas checkpoint file is loaded during this step.

MODIFY : Change Encodings

The MODIFY function is used to modify encodings set using ADADEF DEFINE. At least one of the optional encoding parameters must be specified.

Changing the FACODE, FWCODE, or UWCODE parameters does not affect files already loaded since the actual encoding of their fields is stored in the FCB. The purpose of the database-wide setting is to serve as a default when loading files.

ADADEF MODIFY [**FACODE=**{alpha-EBCDIC-key | current-setting}]
[**FWCODE=**{wide-key | current -setting}]
[**NOUSERABEND**]
[**UACODE=**{alpha-ASCII-key | current-setting}]
[**UES=**{**YES** | **NO**}]
[**UWCODE=**{wide-key | current-setting}]

Optional Parameters

FACODE : Encoding for Alphanumeric Fields

The FACODE parameter specifies the default encoding for alphanumeric fields for all files in the database. The encoding must be derived from EBCDIC encoding; that is, X’40’ is the space character. Modal or “shift” type double-byte character set (DBCS) encodings are supported; fixed type DBCS (DBCS-only) type encodings are not supported. The default encoding key is the current setting.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB. Changing the database-wide setting does not affect files already loaded.

FWCODE : Encoding for Wide-Character Fields

The FWCODE parameter specifies the default encoding for wide-character (W) format fields for all files in the database. The default encoding is the current setting.

The FWCODE parameter can be used to set a wide-character encoding that defines the superset of code points of all user encodings. For example, Unicode encompasses about 50,000 code points as opposed to Host-DBCS and Shift-JIS with about 10,000 code points each.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB. Changing the database-wide setting does not affect files already loaded.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

UACODE : User Encoding for Alphanumeric Fields

The parameter UACODE specifies the default encoding for alpha fields for ASCII users. The encoding must be derived from ASCII encoding; that is, X’20’ is the space character. Encodings for multiple-byte character sets is supported. The default encoding is the current setting.

The UACODE setting is not stored in the loaded file. You can override this encoding for a user session with the OP command.

UES : Universal Encoding Support

The parameter UES can be used to enable or disable universal encoding support for an existing database. Disabling is only possible if no files are loaded with wide-character (W) format fields.

Any valid xxCODE parameter (FACODE, FWCODE, UACODE, UWCODE) implicitly sets UES=YES.

To deactivate UES, you must explicitly set UES=NO.

UWCODE : User Encoding for Wide-Character Fields

The UWCODE parameter specifies the user encoding for wide-character (W) format fields. If the parameter is not specified, the default value is the current setting.

The purpose of the database-wide setting is to serve as a default when loading files. Once loaded, the encoding for a file is stored in its FCB. Changing the database-wide setting does not affect files already loaded.

You can override the default encoding for a user session with the OP command.

Examples

Example 1:

Disable universal encoding support for an existing database. The database contains no files with wide (W) format.

```
ADADEF MODIFY UES=NO
```

Example 2:

Change the default encoding for wide-character (W) format fields for all files in the database from the current setting to code page 835 (traditional Chinese host double byte including 6204 user-defined characters).

ADADEF MODIFY FWCODE=835

Files already loaded are not affected by this change since the actual encoding of their fields is stored in the FCB. The purpose of the database-wide setting is to serve as a default when loading files.

NEWWORK : Defining a Work File

The following parameters are used for Work dataset definition:

ADADEF NEWWORK **WORKSIZE=**size
[**NOUSERABEND**]
[**WORKDEV=**{ device-type | ADARUN-device }]

Notes:

1. *The Adabas nucleus must not be active during this function, and the old Work must be specified in the JCL/JCS.*
2. *The ADADEF NEWWORK function cannot be executed if a pending autorestart exists.*

Essential Parameter

WORKSIZE : Work Dataset Size

The number of blocks or cylinders to be assigned to the Work dataset.

Optional Parameters

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

WORKDEV : Device Type

The device type to be assigned to the new Work dataset.

This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

Example

A new Work dataset is defined with a size of 50 cylinders. The device type is obtained from the ADARUN DEVICE parameter.

```
ADADEF NEWWORK
ADADEF WORKSIZE=50
```

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADADEF with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems and shows examples of each of the job streams.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1	disk	
ADARUN parameters	SYSDTA/DDCARD		*Operations Manual*
ADADEF parameters	SYSDTA/DDKARTE		*Utilities Manual*
ADARUN messages	SYSOUT/DDPRINT		*Messages and Codes*
ADADEF messages	SYSLST/DDDRUCK		*Messages and Codes*

ADADEF JCL Examples (BS2000)

Define Database

In SDF Format:

```
/.ADADEF LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A D E F DEFINE DATABASE
/REMARK *
/ASS-SYSLST L.DEF.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADADEF,DB=yyyyy,IDTNAME=ADABAS5B
ADADEF DEFINE DBNAME=EXAMPLE-DB
ADADEF ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF MAXFILES=120
ADADEF FILE=1,CHECKPOINT
ADADEF NAME= CHECKPOINT ,MAXISN=5000,UISIZE=10B
ADADEF DSSIZE=500B,NISIZE=100B
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADADEF LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A D E F DEFINE DATABASE
/REMARK *
/SYSFILE SYSLST=L.DEF.DEFI
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADADEF,DB=yyyyy,IDTNAME=ADABAS5B
ADADEF DEFINE DBNAME=EXAMPLE-DB
ADADEF ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF MAXFILES=120
ADADEF FILE=1,CHECKPOINT
ADADEF NAME= CHECKPOINT ,MAXISN=5000,UISIZE=10B
ADADEF DSSIZE=500B,NISIZE=100B
/LOGOFF NOSPOOL
```

Define New Work

In SDF Format:

```
/.ADADEF LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A D E F DEFINE NEW WORK
/REMARK *
/ASS-SYSLST L.DEF.NEWW
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADADEF,DB=yyyyy,IDTNAME=ADABAS5B
ADADEF NEWWORK WORKSIZE=60,WORKDEV=dddd
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADADEF LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A D E F DEFINE NEW WORK
/REMARK *
/SYSFILE SYSLST=L.DEF.NEWW
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADADEF,DB=yyyyy,IDTNAME=ADABAS5B
ADADEF NEWWORK WORKSIZE=60,WORKDEV=dddd
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work (Current)	DDWORKR1	disk	
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADADEF parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADADEF messages	DDDRUCK	printer	*Messages and Codes*

ADADEF JCL Examples (OS/390 or z/OS)

Define Database

```
//ADADEF    JOB
//*
//*   ADADEF:
//*      DEFINE THE PHYSICAL LAYOUT OF THE DATABASE
//*      DEFINE THE NUCLEUS SYSTEMFILE: CHECKPOINT FILE
//*
//DEF       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADADEF,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADADEF DEFINE DBNAME=EXAMPLE-DB,DBIDENT=YYYYY
ADADEF        ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF        MAXFILES=120
*
```

```
ADADEF FILE=19,CHECKPOINT
ADADEF   NAME='CHECKPOINT',MAXISN=5000
ADADEF   DSSIZE=100B,NISIZE=3B,UISIZE=3B
/*
//
```

Refer to ADADEF in the MVSJOBS dataset for this example.

Define New Work

```
//ADADEFNW  JOB
//*
//*    ADADEF: DEFINE NEW WORK
//*
//DEF       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADADEF,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADADEF NEWWORK WORKSIZE=60,WORKDEV=eeee
/*
//
```

Refer to ADADEFNW in the MVSJOBS dataset for this example.

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1	disk	
ADARUN parameters	DDCARD	disk/terminal/ reader	Operations Manual
ADADEF parameters	DDKARTE	disk/terminal/ reader	
ADARUN messages	DDPRINT	disk/terminal/ printer	Messages and Codes
ADADEF messages	DDDRUCK	disk/terminal/ printer	

ADADEF JCL Examples (VM/ESA or z/VM)

Define Database

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.ASSO,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDPRINT,DSN=ADADEF.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADADEF.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNDEF.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADADEF.CONTROL,MODE=A
ADARUN
```

Contents of RUNDEF CONTROL A1:

```
ADARUN PROG=ADADEF,DEVICE=dddd,DB=yyyyy
```

Contents of ADADEF CONTROL A1:

```
ADADEF DEFINE DBNAME=EXAMPLE-DB
ADADEF        ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF        MAXFILE=120
*

ADADEF FILE=1,CHECKPOINT
ADADEF NAME='CHECKPOINT',MAXISN=5000,UISIZE=10B
ADADEF                   DSSIZE=500B,NISIZE=100B
```

Define New Work

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.ASSO,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDPRINT,DSN=ADADEF.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADADEF.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNDEF.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADADEF.CONTROL,MODE=A
ADARUN
```

Contents of RUNDEF CONTROL A1:

```
ADARUN PROG=ADADEF,DEVICE=dddd,DB=yyyyy
```

Contents of ADADEF CONTROL A1:

```
ADADEF NEWWORK WORKSIZE=60,WORKDEV=eeee
```

VSE/ESA

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
Work (Current)	WORKR1	disk	*	
ADARUN parameters	– CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADADEF parameters	–	reader	SYSIPT	
ADARUN messages	–	printer	SYSLST	
ADADEF messages	–	printer	SYS009	*Messages and Codes*

***** Any programmer logical unit may be used.

ADADEF JCS Examples (VSE/ESA)

See appendix B for descriptions of the VSE procedures.

Define Database

Refer to member ADADEF.X for this example.

```
* $$ JOB JNM=ADADEF,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADADEF
*       DEFINE THE PHYSICAL LAYOUT OF THE DATABASE
*       DEFINE THE NUCLEUS SYSTEMFILE: CHECKPOINT FILE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADEF,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADADEF DEFINE DBNAME=EXAMPLE-DB,DBIDENT=yyyyy
ADADEF        ASSOSIZE=100,DATASIZE=200,WORKSIZE=40
ADADEF        MAXFILES=120
*
```

```
ADADEF FILE=19,CHECKPOINT
ADADEF   NAME='CHECKPOINT',MAXISN=5000
ADADEF   DSSIZE=100B,NISIZE=3B,UISIZE=3B
/*
/&
* $$ EOJ
```

Define New Work

Refer to member ADADEFNW.X for this example.

```
* $$ JOB JNM=ADADEFNW,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADADEFNW
*       DEFINE NEW WORK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADADEF,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADADEF NEWWORK WORKSIZE=60,WORKDEV=eeee
/*
/&
* $$ EOJ
```

8

ADAFRM : FORMAT

Functional Overview

Primary Adabas direct access (DASD) datasets must be formatted using the ADAFRM utility.

These datasets include the Associator, Data Storage, and Work datasets as well as the intermediate storage (temp, sort, and command/protection/recovery logging) datasets.

Formatting must be performed before any new dataset can be used by the Adabas nucleus or an Adabas utility. After increasing a dataset with the ADADBS INCREASE or ADD function, new RABNs must also be formatted.

ADAFRM also provides functions to reset existing Associator, Data Storage, or Work blocks/cylinders to binary zeros (nulls). Resetting fills the specified blocks in an existing Associator, Data Storage, or Work dataset with binary zeros.

Statement Restrictions

More than one ADAFRM function (ASSOFRM, DATAFRM, RLOGFRM, and so on) can be performed in the same job. However, each function must be specified on separate statements. See the examples at the end of the chapter for more information.

Formatting Operation

Formatting with ADAFRM comprises two basic operations:

1. creating blocks (called RABNS) on the specified tracks/cylinders;
2. filling the created blocks with binary zeros (nulls).

Formatting Modes

There are three ADAFRM formatting modes:

1. Format a **new** dataset (...FRM functions). Only the dataset specified by the function name and the NUMBER parameter is accessed and formatted. The FROMRABN parameter cannot be specified when formatting a new dataset.

2. Format **part of an existing** dataset (ASSOFRM, DATAFRM, WORKFRM, and TEMPFRM functions). Here, the FROMRABN parameter **must** be specified, except on OS/390 and MVS/ESA platforms. When formatting Work and Data Storage (WORKFRM and DATAFRM functions), the ADAFRM job control must also contain the Associator datasets.

   This formatting mode is used in combination with the ADADBS INCREASE function, or to increase a temp dataset that was too small so that an interrupted ADALOD job can be restarted. The logical increase using ADADBS INCREASE must precede the physical increase using ADAFRM. Note that the FROMRABN option is available in this context only under VSE/ESA, VM/ESA or z/VM, and BS2000. See the ADADBS INCREASE examples starting on page 158.

3. Reformat **blocks of an existing** dataset (...RESET functions). This mode opens all Associator, Data Storage, and Work datasets in the database for access. The FROMRABN parameter is **must** be specified for these functions.

Syntax

Format the Associator (ASSO..) or Data Storage (DATA..) dataset:

ADAFRM { **ASSOFRM** | **DATAFRM** } **SIZE=**size
[**DEVICE=**{device-type|ADARUN-device }]
[{ **FROMRABN=**{start-rabn | **NEXT**} |
NUMBER={dataset-number | **1**} }]
[**NOUSERABEND**]

Format the Work (WORK..), command log (CLOG..), protection log (PLOG..), or sort (SORT..) dataset:

ADAFRM { **WORKFRM** | **CLOGFRM** | **PLOGFRM** | **SORTFRM**}
SIZE=size
[**DEVICE=**{device-type|ADARUN-device }]
[{ **FROMRABN=**start-rabn |
NUMBER={dataset-number | **1**} }]
[**NOUSERABEND**]

Format the recovery log (RLOG..) dataset:

ADAFRM RLOGFRM **SIZE=**size
[**DEVICE=**{device-type|ADARUN-device }]
[**NOUSERABEND**]

Format a temp (TEMP..) dataset:

ADAFRM TEMPFRM **SIZE=**size
[**DEVICE=**{device-type|ADARUN-device }]
[**FROMRABN=**start-rabn]
[**NOUSERABEND**]

Reformat blocks of an existing Associator, Data Storage, or Work dataset:

ADAFRM { **ASSORESET** | **DATARESET** | **WORKRESET** }
SIZE=size
FROMRABN=start-rabn
[**NOUSERABEND**]

Essential Parameter

SIZE : Size of Area to be Formatted

SIZE specifies the size of the area to be formatted (or reset). Blocks (a decimal value followed by “B”) or cylinders may be specified. For the RLOGFRM function, the size must be the same as that specified by the RLOGSIZE parameter on the ADARAI utility’s PREPARE function. See volume 2 of the *Adabas Utilities Manual*, page 136.

Optional Parameters

DEVICE : Device Type

DEVICE is the physical device type on which the area to be formatted is contained. If DEVICE is not specified, the device type specified by the ADARUN DEVICE parameter is used.

FROMRABN : Starting RABN

FROMRABN specifies the RABN at which formatting or resetting is to begin. This parameter may only be used for an existing dataset; NUMBER cannot be specified in the same ADAFRM job as FROMRABN.

When FROMRABN is specified with a ...FRM function, formatting begins at the FROMRABN point and continues up to the **highest complete track** before the RABN computed from FROMRABN + SIZE (assuming a size specified in or converted to blocks). This means that the last track within the specified range (FROMRABN + SIZE) will be formatted **only** if all the track’s RABNs are within that range.

When increasing the size of an ASSO or DATA dataset, FROMRABN is available as an option only under VSE/ESA, VM/ESA or z/VM, and BS2000. The specified RABN must be one higher than the highest allocated RABN before the logical increase using ADADBS (which must precede the physical increase using ADAFRM). FROMRABN=NEXT instructs ADAFRM to take the first unformatted RABN as the value for FROMRABN. ADAFRM then verifies that the range of blocks determined for formatting by the NEXT value is contained in the free space table (FST). If not, ADAFRM terminates with ERROR–126. On OS/390, FROMRABN should only be used to reformat existing blocks as the last record pointer in the VTOC cannot be modified by function FROMRABN. See the examples for ADADBS INCREASE starting on page 158.

This parameter is **required** for the ASSORESET, DATARESET and WORKRESET functions. When specified with the function ASSORESET, the FROMRABN value must be greater than 30.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NUMBER : Dataset Number

NUMBER selects the nonsequential command log, nonsequential protection log, Associator, Data Storage and sort dataset to be formatted. The default is 1 (first dataset). Values allowed for

- the Associator (ASSO) or Data Storage (DATA) are 1 through 5;
- protection logs (PLOGs) or command logs (CLOGs) are 2 through 8;
- the recovery log (RLOG) is just 1;
- SORT is either 1 or 2 (1 only on VSE systems); and
- WORK or TEMP is either 1 or the default.

ADAFRM ...FRM function statements cannot specify (and will not default to) a NUMBER value if other ADAFRM statements in the same job specify a FROMRABN value.

NUMBER must match the number suffix of the related data definition (“DD/...”) statement. See the tables of allowed statements and the examples starting on page 257.

Examples

Example 1:

Format 50 cylinders for the Associator, 200 cylinders for Data Storage, 10 cylinders for Work, and 2 cylinders for the recovery log (RLOG).

```
ADAFRM  ASSOFRM SIZE=50,DEVICE=3380
ADAFRM  DATAFRM SIZE=200,DEVICE=3380
ADAFRM  WORKFRM SIZE=10,DEVICE=3380
ADAFRM  RLOGFRM SIZE=2
```

Example 2:

One cylinder for nonsequential command log dataset 1, and 1 cylinder for nonsequential command log dataset 2 are to be formatted.

```
ADAFRM  CLOGFRM SIZE=1,DEVICE=3350,NUMBER=1
ADAFRM  CLOGFRM SIZE=1,DEVICE=3350,NUMBER=2
```

Example 3:

The first two blocks of an existing Work dataset are to be reset to binary zeros.

```
ADAFRM WORKRESET FROMRABN=1,SIZE=2B
```

Example 4:

Assuming the Data Storage dataset is on a 3380 disk (9 blocks/track, 15 tracks/cylinder), 100 cylinders—starting at cylinder position 201 relative to the beginning of the dataset—will be formatted.

```
ADAFRM DATAFRM SIZE=100,FROMRABN=26992
```

Example 5:

Under VSE/ESA, VM/ESA, z/VM, or BS2000, assuming the Associator of the database has just been increased by 200 cylinders, this job formats the new space in the database. For more detailed examples across all supported platforms, see the ADADBS INCREASE examples starting on page 158.

```
ADAFRM ASSOFRM SIZE=200,FROMRABN=NEXT
```

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADAFRM with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems and shows examples of each of the job streams.

Note:
When running with the optional Recovery Aid (RLOG), all temporary datasets must also be cataloged in the job control.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	datasets to be formatted
Data Storage	DDDATARn		
Work	DDWORKR1		
Temp	DDTEMPR1		
Sort	DDSORTRn		
Multiple command logs	DDCLOGRn		
Multiple protection logs	DDPLOGRn		
Recovery log	DDRLOGR1		
ADARUN parameters	SYSDTA/DDCARD		*Operations Manual*
ADAFRM parameters	SYSDTA/DDKARTE		
ADARUN messages	SYSOUT/DDPRINT		*Messages and Codes*
ADAFRM messages	SYSLST/DDDRUCK		*Messages and Codes*

ADAFRM JCL Example (BS2000)

In SDF Format:

```
/.ADAFRM LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A F R M ALL FUNCTIONS
/REMARK *
```

```
/ASS-SYSLST L.FRM
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,OPEN-MODE=OUTIN,BUFF-LEN=STD(1)
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,OPEN-MODE=OUTIN,BUFF-LEN=STD(2)
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,OPEN-MODE=OUTIN,BUFF-LEN=STD(2)
/SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP,OPEN-MODE=OUTIN,BUFF-LEN=STD(2)
/SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT,OPEN-MODE=OUTIN,BUFF-LEN=STD(2)
/SET-FILE-LINK DDPLOGR1,ADAyyyyy.PLOGR1,OPEN-MODE=OUTIN,BUFF-LEN=STD(2)
/SET-FILE-LINK DDPLOGR2,ADAyyyyy.PLOGR2,OPEN-MODE=OUTIN,BUFF-LEN=STD(2)
/SET-FILE-LINK DDRLOGR1,ADAyyyyy.RLOGR1,OPEN-MODE=OUTIN,BUFF-LEN=STD(2)
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAFRM,DB=yyyyy,IDTNAME=ADABAS5B
ADAFRM ASSOFRM SIZE=100
ADAFRM DATAFRM SIZE=200
ADAFRM WORKFRM SIZE=40
ADAFRM SORTFRM SIZE=25
ADAFRM TEMPFRM SIZE=10
ADAFRM PLOGFRM SIZE=40,NUMBER=1
ADAFRM PLOGFRM SIZE=40,NUMBER=2
ADAFRM RLOGFRM SIZE=10
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAFRM LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A F R M ALL FUNCTIONS
/REMARK *
/SYSFILE SYSLST=L.FRM
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,OPEN=OUTIN,BLKSIZE=(STD,1)
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,OPEN=OUTIN,BLKSIZE=(STD,2)
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1,OPEN=OUTIN,BLKSIZE=(STD,2)
/FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1,OPEN=OUTIN,BLKSIZE=(STD,2)
/FILE ADAyyyyy.SORT ,LINK=DDSORTR1,OPEN=OUTIN,BLKSIZE=(STD,2)
/FILE ADAyyyyy.PLOGR1,LINK=DDPLOGR1,OPEN=OUTIN,BLKSIZE=(STD,2)
/FILE ADAyyyyy.PLOGR2,LINK=DDPLOGR2,OPEN=OUTIN,BLKSIZE=(STD,2)
/FILE ADAyyyyy.RLOGR1,LINK=DDRLOGR1,OPEN=OUTIN,BLKSIZE=(STD,2)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADAFRM,DB=yyyyy,IDTNAME=ADABAS5B
ADAFRM ASSOFRM SIZE=100
ADAFRM DATAFRM SIZE=200
ADAFRM WORKFRM SIZE=40
```

```
ADAFRM SORTFRM SIZE=25
ADAFRM TEMPFRM SIZE=10
ADAFRM PLOGFRM SIZE=40,NUMBER=1
ADAFRM PLOGFRM SIZE=40,NUMBER=2
ADAFRM RLOGFRM SIZE=10
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	datasets to be formatted
Data Storage	DDDATARn		
Work	DDWORKR1		
Temp	DDTEMPR1		
Sort	DDSORTRn		
Multiple command logs	DDCLOGRn		
Multiple protection logs	DDPLOGRn		
Recovery log	DDRLOGR1		
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADAFRM parameters	DDKARTE	disk	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAFRM messages	DDDRUCK	printer	*Messages and Codes*

ADAFRM JCL Example (OS/390 or z/OS)

Refer to ADAFRM in the MVSJOBS dataset for this example.

```
//ADAFRM    JOB
//*
//*     ALLOCATE AND FORMAT THE DATABASE COMPONENTS
//*
//*     MORE THAN ONE DATASET CAN BE FORMATTED IN A SINGLE RUN
//*
//*
```

```
//FRM       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD  <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.ASSOR1, <=== ASSO
//            SPACE=(CYL,(0,100)),UNIT=DISK,VOL=SER=VOL001
//DDDATAR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.DATAR1, <=== DATA
//            SPACE=(CYL,(0,200)),UNIT=DISK,VOL=SER=VOL002
//DDWORKR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.WORKR1, <=== WORK
//            SPACE=(CYL,(0,40)),UNIT=DISK,VOL=SER=VOL003
//DDSORTR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.SORTR1, <=== SORT
//            SPACE=(CYL,(0,100)),UNIT=DISK,VOL=SER=VOL003
//DDTEMPR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.TEMPR1, <=== TEMP
//            SPACE=(CYL,(0,100)),UNIT=DISK,VOL=SER=VOL003
//DDPLOGR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.PLOGR1, <=== PLOG1
//            SPACE=(CYL,(50)),UNIT=DISK,VOL=SER=VOL003
//DDPLOGR2  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.PLOGR2, <=== PLOG2
//            SPACE=(CYL,(50)),UNIT=DISK,VOL=SER=VOL003
//DDCLOGR1  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.CLOGR1, <=== CLOG1
//            SPACE=(CYL,(50)),UNIT=DISK,VOL=SER=VOL003
//DDCLOGR2  DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.CLOGR2, <=== CLOG2
//            SPACE=(CYL,(50)),UNIT=DISK,VOL=SER=VOL003
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAFRM,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAFRM ASSOFRM SIZE=100,DEVICE=dddd
ADAFRM DATAFRM SIZE=200,DEVICE=dddd
ADAFRM WORKFRM SIZE=40,DEVICE=dddd
ADAFRM SORTFRM SIZE=100,DEVICE=dddd
ADAFRM TEMPFRM SIZE=100,DEVICE=dddd
ADAFRM PLOGFRM SIZE=50,NUMBER=1,DEVICE=dddd
ADAFRM PLOGFRM SIZE=50,NUMBER=2,DEVICE=dddd
ADAFRM CLOGFRM SIZE=50,NUMBER=1,DEVICE=dddd
ADAFRM CLOGFRM SIZE=50,NUMBER=2,DEVICE=dddd
/*
//
```

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	datasets to be formatted
Data Storage	DDDATARn		
Work	DDWORKR1		
Temp	DDTEMPR1		
Sort	DDSORTRn		
Multiple command logs	DDCLOGRn		
Multiple protection logs	DDPLOGRn		
Recovery log	DDRLOGR1		
ADARUN parameters	DDCARD	disk/terminal/reader	*Operations Manual*
ADAFRM parameters	DDKARTE	disk/terminal/reader	
ADARUN messages	DDPRINT	disk/terminal/printer	*Messages and Codes*
ADAFRM messages	DDDRUCK	disk/terminal/printer	*Messages and Codes*

ADAFRM JCL Example (VM/ESA or z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDPLOGR1,DSN=ADABASVv.PLOG1,VOL=PLOGV1
DATADEF DDPLOGR2,DSN=ADABASVv.PLOG2,VOL=PLOGV2
DATADEF DDRLOGR1,DSN=ADABASVv.RLOG1,VOL=RLOGV1
DATADEF DDPRINT,DSN=ADAFRM.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAFRM.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNFRM.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAFRM.CONTROL,MODE=A
ADARUN
```

Contents of RUNFRM CONTROL A1:

```
ADARUN PROG=ADAFRM,DEVICE=dddd,DB=yyyyy
```

Contents of ADAFRM CONTROL A1:

```
ADAFRM ASSOFRM SIZE=100
ADAFRM DATAFRM SIZE=200
ADAFRM WORKFRM SIZE=40
ADAFRM SORTFRM SIZE=25
ADAFRM TEMPFRM SIZE=10
ADAFRM PLOGFRM SIZE=40
ADAFRM PLOGFRM SIZE=40,NUMBER=2
ADAFRM RLOGFRM SIZE=10
```

VSE/ESA

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	files to be formatted
Data Storage	DATARn			
Work	WORKR1			
Temp	TEMPR1			
Sort	SORTR1			
Multiple command log	CLOGRn			
Multiple protection log	PLOGRn			
Recovery log	RLOGR1			
ADARUN parameters	– CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAFRM parameters	–	reader	SYSIPT	
ADARUN messages	–	printer	SYSLST	*Messages and Codes*
ADAFRM messages	–	printer	SYS009	*Messages and Codes*

***** Any programmer logical unit may be used.

ADAFRM JCS Example (VSE/ESA)

See appendix B for descriptions of the VSE/ESA procedures (PROCs).

Refer to member ADAFRM.X for this example.

```
* $$ JOB JNM=ADAFRM,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAFRM
*       FORMAT THE DATABASE COMPONENTS
?/ EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAFRM,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAFRM ASSOFRM SIZE=100,DEVICE=dddd
ADAFRM DATAFRM SIZE=200,DEVICE=dddd
ADAFRM WORKFRM SIZE=40,DEVICE=dddd
ADAFRM SORTFRM SIZE=100,DEVICE=dddd
ADAFRM TEMPFRM SIZE=100,DEVICE=dddd
ADAFRM PLOGFRM SIZE=50,NUMBER=1,DEVICE=dddd
ADAFRM PLOGFRM SIZE=50,NUMBER=2,DEVICE=dddd
ADAFRM CLOGFRM SIZE=50,NUMBER=1,DEVICE=dddd
ADAFRM CLOGFRM SIZE=50,NUMBER=2,DEVICE=dddd
/*
/&
* $$ EOJ
```

9

ADAICK : CHECK INDEX AND ADDRESS CONVERTER

Functional Overview

ADAICK checks the physical structure of the Associator. This includes validating the index based upon the descriptor value structures and the Associator extents defined by the general control block (GCB) and file control block (FCB).

The ADAICK utility should be used only for diagnostic purposes.

ADAICK can perform the following functions:

- Check index and address converter for specific files;
- Print/dump the contents of any ASSO or DATA block in the database;
- Print/dump the contents of normal (NI) and upper (UI) indexes.
- Print/dump formatted the contents of GCBs, FCBs, FDTs, and PPTss.

Notes:

1. *ADAICK can run with or without an active Adabas nucleus.*
2. *A pending autorestart condition is ignored.*
3. *If the nucleus is active, ADAICK synchronizes its operation with the active nucleus unless the NOOPEN parameter is specified.*

Summary of Functions

Function	Description	Page
ACCHECK	check the address converter	267
ASSOPRINT	print/dump Associator blocks	268
BATCH	set printout width to 132 characters per line	269
DATAPRINT	print/dump Data Storage blocks	270
DSCHECK	print/dump Data Storage record	271
DUMP	suspend dump suppression set using NODUMP function	272
FCBPRINT	print/dump file control block	273
FDTPRINT	print/dump file definition table	274
GCBPRINT	print/dump general control block	275
ICHECK	check index and address converter	276
INT	cancel formatted printout suppression set using NOINT function	277
NIPRINT	print/dump normal index	278
NOBATCH	set printout width to 80 characters per line	279
NODUMP	suppress dumps	280
NOINT	suppress formatted printout	281
PPTPRINT	print/dump the parallel participant table (PPT)	282
UIPRINT	print/dump upper index	284

ACCHECK : Check Address Converter

ADAICK ACCHECK **FILE=**file-number
[**NOOPEN**]
[**NOUSERABEND**]

Essential Parameter

FILE : File to be Checked

The file to be checked. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file used by ADAICK is checked.

Optional Parameters

NOOPEN : Prevent Open Synchronization

When starting, ADAICK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAICK from issuing the open call.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

ASSOPRINT : Print/Dump Associator Blocks

ADAICK ASSOPRINT **RABN=**{rabn | rabn – rabn}
[**NOUSERABEND**]

Essential Parameter

RABN : RABNs to be Processed

The RABN (or a single range of RABNs) to be printed/dumped. If ADAICK can determine the type of information stored in the block (for example. UI, NI,...), it produces a formatted printout.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

BATCH : Set Printout Width to 132 Characters Per Line

ADAICK BATCH [**NOUSERABEND**]

If ADAICK is to be used in batch mode, this function may be used to set the printout width to 132 characters per line. See the NOBATCH function on page 279 for information about resetting the printout width.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

DATAPRINT : Print/Dump Data Storage Blocks

ADAICK DATAPRINT **RABN=**{rabn | rabn – rabn}
[**NOUSERABEND**]

Essential Parameter

RABN : RABNs to be Processed

The RABN (or a single range of RABNs) to be printed/dumped.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

DSCHECK : Print/Dump Content of Data Storage Record

```
ADAICK DSCHECK  FILE=file-number
                [ISN=isn-of-record]
                [NOOPEN]
                [NOUSERABEND]
```

Essential Parameter

FILE : File Number

The number of the file for which the record is to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameters

ISN : ISN of Data Storage Record

The ISN of the Data Storage record to be printed. If ISN is omitted, the DSCHECK function prints the last ISN plus 1.

NOOPEN : Prevent Open Resynchronization

When starting, ADAICK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAICK from issuing the open call.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

DUMP : Suspend Dump Suppression

ADAICK DUMP [**NOUSERABEND**]

This function suspends suppression of ADAICK dumps. See the NODUMP function on page 280 for information about suppressing dumps.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

FCBPRINT : Print/Dump File Control Block

ADAICK FCBPRINT **FILE=**file-number
[**NOOPEN**]
[**NOUSERABEND**]

The file control block (FCB) is to be dumped/printed.

Essential Parameter

FILE : File Number

The number of the file for which the FCB is to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameters

NOOPEN : Prevent Open Resynchronization

When starting, ADAICK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAICK from issuing the open call.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

FDTPRINT : Print/Dump Field Definition Table

ADAICK FDTPRINT **FILE=**file-number
[**NOUSERABEND**]

The field definition table (FDT) is to be dumped/printed.

Essential Parameter

FILE : File Number

The number of the file for which the FDT is to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameters

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

GCBPRINT : Print/Dump General Control Block

ADAICK GCBPRINT [**NOUSERABEND**]

The general control block (GCB) is to be dumped/printed.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

ICHECK : Check Index and Address Converter

ADAICK ICHECK **FILE=**{file-number | file-number – file-number}
[**NOOPEN**]
[**NOUSERABEND**]

Essential Parameter

FILE : Files to be Checked

The specified file (or a single range of files) to be checked. FILE must be specified.

Optional Parameters

NOOPEN : Prevent Open Resynchronization

When starting, ADAICK normally performs a utility open call to the nucleus to assure that no blocks of the affected file or files are still in the nucleus buffer pool. However, this also locks the file for other users. Specifying NOOPEN prevents ADAICK from issuing the open call.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

INT : Cancel Formatted Printout Suppression

ADAICK INT [**NOUSERABEND**]

This function cancels suppression of the formatted printout produced by ADAICK. See the NOINT function on page 281 for information about suppressing formatted printouts.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NIPRINT : Print/Dump Normal Index

ADAICK NIPRINT **FILE=**file-number
[**NOUSERABEND**]

Essential Parameter

FILE : File Number

The number of the file for which the normal index is to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NOBATCH : Set Print Width to 80 Characters Per Line

ADAICK NOBATCH [**NOUSERABEND**]

The printout width is set to 80 characters per line. See the BATCH function on page 269 for information about resetting the printout width.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NODUMP : Suppress Dumps

ADAICK NODUMP [**NOUSERABEND**]

This function suppresses ADAICK dumps. See the DUMP function on page 272 for information about suspending the suppression.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NOINT : Suppress Formatted Printout

ADAICK NOINT [**NOUSERABEND**]

This function suppresses the formatted printout produced by ADAICK. See the INT function on page 277 for information about suspending the suppression.

Optional Parameter

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PPTPRINT : Print/Dump Parallel Participant Table

ADAICK PPTPRINT [**NOUSERABEND**]

The parallel participant table (PPT) for the Adabas cluster is to be dumped/printed. Note that in the dump/print, ‘PPH’ is the tag for the PPT header and ‘PPE’ is the tag for the PPT entries.

Each of the 32 blocks (RABNs) allocated for the PPT represents a single nucleus in the cluster and comprises

- a single header of fixed length; and
- multiple entries of variable length.

In the dump/print, ‘PPH’ is the tag for a PPT block’s header and ‘PPE’ is the tag for a PPT block’s entries.

Optional Parameters

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

Example Output

```
ADAICK PPTPRINT

   MEANING: DUMP ASSO BLOCK 000000BF THRU 000000DE
DB 00072 PPT AT RABN      000000BF
DB 00072 PPT BLOCK NUMBER 01
DB 00072 PPH+000                 NUMBER OF ENTRIES: 03
DB 00072 PPH+001                 NUCLEUS INDICATOR: C0
DB 00072 PPH+002                    EXTERNAL NUCID: 0000
DB 00072 PPH+004                           UNUSED: 00000000
DB 00072 PPE+000               LENGTH OF PPT ENTRY: 0023
DB 00072 PPE+002 HDDATE FROM FIRST PLOG BLK (HIGH): 00000000
DB 00072 PPE+006  HDDATE FROM FIRST PLOG BLK (LOW): 00000000
DB 00072 PPE+00A                   PTT STATUS FLAG: 00
DB 00072 PPE+00B                   ID OF PPT ENTRY: W
DB 00072 DATASET=ADABAS.GB.UTI.72.WORKR1
DB 00072 PPE+000               LENGTH OF PPT ENTRY: 0023
DB 00072 PPE+002 HDDATE FROM FIRST PLOG BLK (HIGH): 00000000
DB 00072 PPE+006  HDDATE FROM FIRST PLOG BLK (LOW): 00000000
DB 00072 PPE+00A                   PTT STATUS FLAG: 00
DB 00072 PPE+00B                   ID OF PPT ENTRY: 1
DB 00072 DATASET=ADABAS.GB.UTI.72.PLOGR1
DB 00072 PPE+000               LENGTH OF PPT ENTRY: 0023
DB 00072 PPE+002 HDDATE FROM FIRST PLOG BLK (HIGH): 00000000
DB 00072 PPE+006  HDDATE FROM FIRST PLOG BLK (LOW): 00000000
DB 00072 PPE+00A                   PTT STATUS FLAG: 00
DB 00072 PPE+00B                   ID OF PPT ENTRY: 2
DB 00072 DATASET=ADABAS.GB.UTI.72.PLOGR2

ASSO BLOCK 000000BF  PPT
0000  03C00000 00000000  00230000 00000000  *.ù       .      *
0010  000000E6 7AC1C4C1  7A5BC7C5 C24BE4E3  *   WADABAS.GB.UT*
0020  C94BF7F2 4BE6D6D9  D2D9F100 23000000  *I.74.WORKR1 .   *
0030  00000000 0000F17A  C1C4C17A 5BC7C5C2  *      1ADABAS.GB*
0040  4BE4E3C9 4BF7F24B  D7D3D6C7 D9F10023  *.UTI.74.PLOGR1 .*
0050  00000000 00000000  00F27AC1 C4C17A5B  *         2ADABAS*
0060  C7C5C24B E4E3C94B  F7F24BD7 D3D6C7D9  *.GB.UTI.74.PLOGR*
0070  F2000000 00000000  00000000 00000000  *2               *
0080  00000000 00000000  00000000 00000000  *                *
      SAME
0FF0  00000000 00000000  00000000          *            *

DB 00072 PPT RABNS 000000C0 - 000000DE (02-32) ARE UNUSED

A D A I C K  TERMINATED NORMALLY                          2000-07-26  09:45:19
```

UIPRINT : Print/Dump Upper Index

ADAICK UIPRINT **FILE=**file-number
[**NOUSERABEND**]

Essential Parameter

FILE : File Number

The number of the file for which the upper index(es) is/are to be printed/dumped. A file number is required the first time you execute ADAICK.

If FILE is omitted on subsequent executions, the last file accessed by ADAICK is used.

Optional Parameters

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

Examples

Example 1:

Check the index and address converter for file 18 and print/dump the FDT for this file.

```
ADAICK ICHECK FILE=18
ADAICK FDTPRINT
```

Example 2:

Set printout width to 120 characters per line (printer). Check index and address converter for file 1 and print/dump Associator RABNs 123 through 135.

```
ADAICK BATCH
ADAICK ICHECK  FILE=1
ADAICK ASSOPRINT RABN=123–135
```

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADAICK with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems and shows examples of each of the job streams.

Collation with User Exit

If a collation user exit is to be used during ADAICK execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding subsystem (UES), the format of the collation descriptor user exit parameter is

ADARUN CDXnn**=**exit-name

—where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01–08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *DBA Reference Manual* for more information.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	SYSDTA/ DDCARD		*Operations Manual*
ADAICK parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT DDPRINT		*Messages and Codes*
ADAICK messages	SYSLST DDDRUCK		*Messages and Codes*

ADAICK JCL Example (BS2000)

In SDF Format:

```
/.ADAICK LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK *A D A I C K INDEX CHECK
/REMARK *
/REMARK *
/ASS-SYSLST L.ICK.DATA
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyyy.DATA,SHARE-UPD=YES
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAICK,DB=yyyyy,IDTNAME=ADABAS5B
ADAICK ICHECK FILE=27
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAICK LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK *A D A I C K INDEX CHECK
/REMARK *
/REMARK *
/SYSFILE SYSLST=L.ICK.DATA
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADAICK,DB=yyyyy,IDTNAME=ADABAS5B
ADAICK ICHECK FILE=27
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADAICK parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAICK messages	DDDRUCK	printer	*Messages and Codes*

ADAICK JCL Example (OS/390 or z/OS)

Refer to ADAICK in the MVSJOBS dataset for this example.

```
//ADAICK    JOB
//*
//*   ADAICK:
//*       INDEX AND ADDRESS CONVERTER CHECK
//*
//ICK       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAICK,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAICK ICHECK FILE=1-3
/*
//
```

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
ADARUN parameters	DDCARD	disk/terminal/reader	*Operations Manual*
ADAICK parameters	DDKARTE	disk/terminal/reader	
ADARUN messages	DDPRINT	disk/terminal/printer	*Messages and Codes*
ADAICK messages	DDDRUCK	disk/terminal/printer	*Messages and Codes*

ADAICK JCL Example (VM/ESA or z/VM)

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDPRINT,DSN=ADAICK.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAICK.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNICK.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAICK.CONTROL,MODE=A
ADARUN
```

Contents of RUNICK CONTROL A1:

```
ADARUN PROG=ADAICK,DEVICE=dddd,DB=yyyyy
```

Contents of ADAICK CONTROL A1:

```
ADAICK ICHECK FILE=27
```

VSE/ESA

File	Symbolic Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Data Storage	DATARn	disk	*	
ADARUN parameters	— CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAICK parameters		reader	SYSIPT	
ADARUN messages		printer	SYSLST	Messages and Codes
ADAICK messages		printer	SYS009	Messages and Codes

***** Any programmer logical unit may be used.

ADAICK JCS Example (VSE/ESA)

See appendix B for descriptions of the VSE/ESA procedures (PROCs).

Refer to member ADAICK.X for this example.

```
* $$ JOB JNM=ADAICK,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAICK
*       INDEX AND ADDRESS CONVERTER CHECK
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAICK,MODE=SINGLE,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAICK ICHECK FILE=1-3
/*
/&
* $$ EOJ
```

10

ADAINV : INVERT

Functional Overview

The ADAINV utility is used to

Function	Action	Page
COUPLE	couple two files	292
INVERT	create a descriptor	298

The INVERT function

- modifies the field definition table (FDT) to indicate that the specified field is a descriptor; and
- adds all values and corresponding ISN lists for the field to the inverted list.

The newly defined descriptor may then be used in the same manner as any other descriptor. This function may also be used to create a subdescriptor, superdescriptor, phonetic descriptor, or hyperdescriptor.

The COUPLE function adds a common descriptor to two files (updates their inverted lists). Any two files may be coupled provided that a common descriptor with identical format and length definitions is present in both files. A single file may be coupled with up to 18 other files, but only one coupling relationship may exist between any two files at any one time. A file may not be coupled to itself.

Note:
Only files with numbers 255 or lower can be coupled.

Changes affecting a coupled file's inverted lists are automatically made to the other file. The DBA should consider the additional overhead required to update the coupling lists when the descriptor used as the basis for coupling is updated, or when records are added to or deleted from either file. If a field that is not defined with the NU option is used as the basis for coupling and the field contains a large number of null values, a considerable amount of additional execution time and required disk space to store the coupling lists may result.

An interrupted ADAINV operation can be restarted without first having to restore the file.

COUPLE : Define a File-Coupling Descriptor

ADAINV COUPLE **FILES=**file-number1, file-number2
DESCRIPTOR='fieldname, fieldname'
SORTSIZE=size
TEMPSIZE=size
[**LPB=**{ prefetch-buffer-size | ADARUN-lu *}*]
[**LWP=**{ workpool-size | **1048576** }]
[**NOUSERABEND**]
[**PASSWORD=***'password'*]
[**SORTDEV=**{ device-type | ADARUN-device *}*]
[**TEMPDEV=**{ device-type | ADARUN-device *}*]
[**TEST**]

Essential Parameters

DESCRIPTOR : Descriptors Used as Basis for Coupling

The DESCRIPTOR parameter defines one descriptor in each file to provide the basis for coupling the files. Subdescriptors or superdescriptors may also be used, or may be defined as or derived from a multiple-value field. The descriptors specified may not be contained within a periodic group, nor be derived from a periodic group. The descriptors can have different names, but must have the same length and format definitions.

FILES : Files to Be Coupled

FILES specifies the two files to be coupled. The number of each file must be 255 or lower. The files specified may not be currently coupled to each other.

SORTSIZE : Sort Size

SORTSIZE specifies the space available for the sort dataset or datasets R1/2 (SORTR2 is not supported under VSE). The value can be either cylinders (a numeric value only) or blocks (a numeric value followed by “B”). If blocks are specified, they should be equivalent to a full number of cylinders. The SORTSIZE parameter must be specified. Refer to the *Adabas DBA Reference Manual* for more information on estimating the sort space.

TEMPSIZE : Temporary Storage Size

TEMPSIZE defines the space available for the temp dataset. The value may be in cylinders (a numeric value only) or blocks (a numeric value followed by “B”). This parameter must be specified.

Optional Parameters

LPB : Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32760 bytes. The default depends on the ADARUN LU parameter; ADAINV may also reduce a specified LPB value if the LU value is too small.

LWP : Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a “K”. If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADAINV run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the sort dataset, a smaller LWP value should be specified when defining descriptors for relatively small files.

The minimum work pool size depends on the sort dataset’s device type:

Sort Device	Minimum LWP Bytes	Minimum LWP Kilobytes
2000	106496	104K
2314	090112	88K
3375	131072	128K
3380	139264	136K
3390	159744	156K

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

If one or both of the files being coupled is security protected, a valid password for the file (or files) must be specified with this parameter. If both files are password-protected, both must have the same password.

SORTDEV : Sort Device Type

ADAINV uses the sort dataset to sort descriptor values. The SORTDEV parameter indicates the device type to be used for the sort dataset. This parameter is required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter. See the MVS job control information on page 307 for specific SORTDEV considerations.

TEMPDEV : Temporary Storage Device Type

ADAINV uses the temp dataset to store intermediate data. The TEMPDEV parameter indicates the device type to be used for this dataset. This parameter is required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Example

```
ADAINV COUPLE FILES=3,4,DESCRIPTOR='AA,BB'
```

Files 3 and 4 are to be coupled. Descriptor AA from file 3 and descriptor BB from file 4 are to be used as the basis for the coupling.

Temporary Space for File Coupling

An intermediate dataset is generated for **each** of the files being coupled.

An entry is written to the dataset for each record contained in the file. Each entry contains the ISN of the record (3 or 4 bytes, depending on the ISNSIZE defined for the files) and the value (in compressed form) of the descriptor being used as the basis for the coupling. If the descriptor is defined with the NU option, no entries are written for records in which the descriptor contains a null value. If the descriptor is a multiple-value field, an entry is written for each different value.

The space required for **each** of the intermediate datasets is a function of the number of records contained in each Adabas file, and the length and the number of the different values present for the coupling descriptor in each record.

Use the following equation to determine the space needed for an intermediate dataset:

SP = RECS • UV • (ISNSIZE + (AVLEN • 4)

—where

SP	intermediate dataset space required (in bytes).
RECS	number of records contained in the coupled file.
UV	average number of unique values per record for the descriptor. If the descriptor is not defined with the NU option, UV is equal to or less than 1. If the descriptor is defined with the NU option, UV is equal to the average number of values per record minus the percentage of records that contain a null value. For example, if the average number of values per record is 1 and 20 percent of the values are null, UV is equal to 1 – 0.2 = 0.8.
ISNSIZE	length of ISNs in the file (3 or 4 bytes).
AVLEN	average length (after compression) of each value for the descriptor.

Example: Calculating Intermediate Space Requirements for File Coupling

The file being coupled has 3-byte ISNs and contains 50,000 records. The descriptor being used as the basis for coupling contains 1 value per record (with no null values) and has an average value length of 5 bytes.

SP = 50,000 • 1 • (3 + (5 + 1))
SP = 50,000 • 9
SP = 450,000 bytes

Associator Coupling Lists

ADAINV matches the two lists, sorts each resulting list, and writes each list to the Associator coupling lists.

The temp dataset stores the matched (coupled) ISNs for each file. An entry is written to the temp dataset for each match found. The entry contains the ISN of each record containing a matching value.

ADAINV sorts the entries stored on the temp dataset using the sort area and writes the sorted entries to the Associator coupling lists for file A. The same process is then repeated for file B.

The temp area size requirement depends on the number of matching values in the two files for the descriptor used to couple the files. Each match requires 6 or 8 bytes, depending on the ISNSIZE defined for the files.

The sort area generally requires twice the amount of space as that needed for the temp area.

File coupling is bidirectional rather than hierarchical in that two coupling lists are created with each list containing the ISNs which are coupled to the other file.

Example: Coupling Lists

Assume that 2 files containing the descriptors AA and BB, respectively, are to be coupled. The values for the first five records of each file are as follows:

File A		File B	
ISN	**Field AA value**	**ISN**	**Field BB value**
1	20	1	18
2	25	2	40
3	27	3	25
4	30	4	20
5	40	5	20

If the two files were coupled using AA and BB as the basis for the coupling, the resulting coupling lists would be:

File A			File B		
ISN in FILE B*	**COUNT**	**COUPLED ISNs**	**ISN in FILE A***	**COUNT**	**COUPLED ISNs**
2	1	5	1	2	4,5
3	1	2	2	1	3
4	1	1	5	1	2
5	1	1			

* Internally, Adabas uses this field like a descriptor to determine the number and the ISNs of the coupled records.

Space for Coupling Lists

The total space requirement for the coupling lists depends upon the number of common values that exist between the two descriptors used as the basis for the coupling.

The space requirement for **each common value** may be estimated as follows:

SP = 4a + 4b + 6ab

—where

SP space requirement for one common value (in bytes);

a number of records in file A containing the common value;

b number of records in file B containing the common value.

The total coupling list requirement is the sum of the space requirements of each common value.

Using sample files A and B as previously defined, space requirements per common value are

Common Value	Space Requirements
20	SP = 4(1) + 4(2) + 6(1 • 2) = 24 bytes
25	SP = 4(1) + 4(1) + 6(1 • 1) = 14 bytes
40	SP = 4(1) + 4(1) + 6(1 • 1) = 14 bytes

Total space required = 24 + 14 + 14 = 52 bytes

Example: Coupling List Space Requirements

Assume that 2 files are being coupled on the field ID. The values for ID are unique within each file. There are 5,000 common values in the coupled files.

Common Value	Space Requirements
n	SP = 4(1) + 4(1) + 6(1) SP = 14 bytes for one common value

There are 5,000 common values, each of which requires 14 bytes. The total space requirement for the coupling lists is 70,000 bytes.

Space Allocation

The coupling lists constructed by ADAINV are contained within the normal (NI) and upper (UI) index for each file being coupled. If the NI or UI component's logical extents currently allocated to the file are used up during ADAINV execution, ADAINV attempts to allocate an additional extent to the component. The size of the extent allocated is equal to 25 percent of the current total size of all logical extents currently assigned to the component. If insufficient space is available or if the maximum of five extents has already been allocated to the component, ADAINV terminates with an error message.

INVERT : Create a Descriptor

The INVERT function creates descriptors, subdescriptors, superdescriptors, hyperdescriptors, phonetic descriptors or collation descriptors for existing fields in a file. Several descriptors may be created in a single ADAINV INVERT run, but only for a single file.

ADAINV INVERT **FILE=** file-num
SORTSIZE= size
TEMPSIZE= size
[**FIELD=** 'field-name [{,option}...]']...
[**COLDE=** 'num,name [,**UQ** [,**XI**]]**=** parent-field']
[**HYPDE=** 'num,name,length,format [{,option}...]**=** {parent-field}**,...**']
[**PHONDE=** 'name (field-name)']
[**SUBDE=** 'name [,**UQ** [,**XI**]]**=** parent-field (begin,end) ']
[**SUPDE=** 'name [,**UQ** [,**XI**]]**=** {parent-field (begin,end) },... ']
[**CODE=** cipher-code]
[**LPB=** {prefetch-buffer-size | ADARUN-lu }]
[**LWP=** {work-pool-size | **1048576** }]
[**NOUSERABEND**]
[**PASSWORD=** 'password']
[**SORTDEV=** {device-type | ADARUN-device }]
[**TEMPDEV=** {device-type | ADARUN-device }]
[**TEST**]

Essential Parameters

FILE : File Number

FILE specifies the file in which the descriptor(s) to be created is contained.

SORTSIZE : Sort Size

SORTSIZE specifies the space available for the sort dataset or datasets R1/2 (SORTR2 is not supported under VSE). The value can be either cylinders (a numeric value only) or blocks (a numeric value followed by “B”). If blocks are specified, they should be equivalent to a full number of cylinders. The SORTSIZE parameter must be specified. Refer to the *Adabas DBA Reference Manual* for more information on estimating the sort space.

TEMPSIZE : Temporary Storage Size

TEMPSIZE defines the space available for the temp dataset. The value may be in cylinders (a numeric value only) or blocks (a numeric value followed by “B”). This parameter must be specified.

Optional Parameters and Subparameters

CODE : Cipher Code

If the file specified with the FILE parameter is ciphered, an appropriate cipher code must be supplied using the CODE parameter.

FIELD / COLDE / HYPDE / PHONDE / SUBDE / SUPDE : Define Descriptor(s)

These parameters may be used to define various types of descriptors. You must specify at least one descriptor definition for the file specified; you may specify more than one descriptor or type of descriptor.

Use the FIELD parameter to define one or more fields as descriptors; use the COLDE parameter for a collation descriptor; HYPDE parameter for a hyperdescriptor; PHONDE for a phonetic descriptor; SUBDE for a subdescriptor; and SUPDE for a superdescriptor.

If provided, a FIELD specification must come before any collation descriptor, hyper-, super-, sub-, or phonetic descriptor specification.

FIELD specifies an existing field (or fields) to be inverted. The field may be an elementary or multiple-value field and may be contained within a periodic group (unless the field is defined with the FI option).

If the descriptor is to be unique, specify “UQ” following the field name. If the uniqueness of the descriptor is to be determined with the index (occurrence number) excluded, specify “XI” as well.

Note:
For Adabas expanded files, ADAINV can only detect unique descriptor violations within the specified component file. If an identical value exists for a unique descriptor in one of the other component files, ADAINV cannot detect it. You must therefore ensure that unique descriptor values remain unique throughout an expanded file.

Although multiple fields can be specified for inversion using the FIELD parameter, only one collation descriptor, hyper-, sub-, super-, or phonetic descriptor is defined per instance of its parameter. Multiple instances of the parameters are allowed per execution of ADAINV.

When inverting a sub- or superfield, the respective SUBDE or SUPDE parameter must specify the same parent fields that were specified when the field was created; otherwise, an error occurs. Begin and end values are taken from the original field definitions.

If a parent field with the NU option is specified, no entries are made in the inverted list for those records containing a null value for the field. For super- and hyperdescriptors, this is true regardless of the presence or absence of values for other descriptor elements.

If a parent field is not initialized and logically falls past the end of the physical record, the inverted list entry for that record is not generated for performance reasons. To generate the inverted list entry in this case, it is necessary to unload short, decompress, and reload the file; or use an application program to initialize the field for each record of the file.

See the ADACMP utility description on page 66 for detailed information about the individual descriptor syntax, subparameter values, and coding.

LPB : Prefetch Buffer Size

LPB specifies the size, in bytes, of the internal prefetch buffer. The maximum value is 32,760 bytes. The default depends on the ADARUN LU parameter; ADAINV may also reduce a specified LPB value if the LU value is too small.

LWP : Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a "K". If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADAINV run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the Sort dataset, a smaller LWP value should be specified when defining descriptors for relatively small files.

The minimum work pool size depends on the Sort dataset's device type:

Sort Device	Minimum LWP Bytes	Minimum LWP Kilobytes
2000	106496	104K
2314	090112	88K
3375	131072	128K
3380	139264	136K
3390	159744	156K

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

PASSWORD : File Password

If the file specified with the FILE parameter is security protected, the file's password must be supplied using this parameter.

SORTDEV : Sort Device Type

ADAINV uses the sort dataset to sort descriptor values. The SORTDEV parameter indicates the device type to be used for the sort dataset. This parameter is required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter. See the MVS job control information at the end of this chapter for specific MVS SORTDEV considerations.

TEMPDEV : Temporary Storage Device Type

ADAINV uses the temp dataset to store intermediate data. The TEMPDEV parameter indicates the device type to be used for this dataset. This parameter is required only if the device type to be used is different from that specified with the ADARUN DEVICE parameter.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

Space Allocation for the INVERT Function

The values for the field being inverted and the ISNs of the records containing the values are written to the inverted list (normal and upper indexes).

If either the normal or upper index logical extent is exhausted during ADAINV execution, ADAINV allocates an additional extent. The size of the extent allocated is equal to 25 percent of the current total size of all the normal index extents currently allocated to the file.

If sufficient space is not available for the new extent or if the maximum of five extents has already been allocated, ADAINV terminates with an error message.

Examples

Example 1:

```
ADAINV INVERT FILE=3,FIELD='AR',TEMPSIZE=10,SORTSIZE=5
```

Field AR in file 3 is to be made a descriptor.

Example 2:

```
ADAINV INVERT FILE=5,SUBDE='SA=AA(1,4)'
ADAINV            TEMPSIZE=6,SORTSIZE=3
```

Subdescriptor SA is to be created using field AA (positions 1-4) in file 5 as the parent field.

Example 3:

```
ADAINV INVERT FILE=6,SUPDE='SB=AA(1,4),AB(1,1)'
ADAINV            TEMPSIZE=5,SORTSIZE=3
```

Superdescriptor SB is to be created using fields AA (positions 1-4) and AB (position 1) in file 6.

Example 4:

```
ADAINV INVERT FILE=1,PHONDE='XX(AA)'
ADAINV            TEMPSIZE=6,SORTSIZE=3
```

A phonetic descriptor XX is created using field AA as the source field.

Example 5;

```
ADAINV INVERT FILE=6,COLDE='1,Y1=AA'
ADAINV            TEMPSIZE=5,SORTSIZE=4
```

Collation descriptor CDX=01 named Y1 is created using AA as the source field.

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADAINV with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems and shows examples of each of the job streams.

Collation with User Exit

If a collation user exit is to be used during ADAINV execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding support (UES), the format of the collation descriptor user exit parameter is

ADARUN CDXnn**=**exit-name

—where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01–08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *Adabas DBA Reference Manual* for more information.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Intermediate storage	DDTEMPR1	disk	
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	When using large files, the Sort area should be split across two volumes.*
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the recovery log option
ADARUN parameters	SYSDTA/DDCARD		Operations Manual
ADAINV parameters	SYSDTA/DDKARTE		
ADARUN messages	SYSOUT/DDPRINT		Messages and Codes
ADAINV messages	SYSLST/DDDRUCK		Messages and Codes

***** Performance can be improved when sorting large files if the sort dataset is split across two volumes (see the BS2000 information in appendix A of the *Operations Manual*). If two datasets are specified, they must both be on the same device type (SORTDEV parameter), and each must be exactly half the size specified with the SORTSIZE parameter.

ADAINV JCL Examples (BS2000)

Couple Files

In SDF Format:

```
/.ADAINV LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A I N V COUPLE FIELD (REFLECTIVE)
/REMARK *
/ASS-SYSLST L.INV.COUP
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
/SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAINV,DB=yyyyy,IDTNAME=ADABAS5B
ADAINV COUPLE FILE=1,3,DESCRIPTOR= AA,AA
ADAINV TEMPSIZE=100,SORTSIZE=50
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAINV LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A I N V COUPLE FIELD (REFLECTIVE)
/REMARK *
/SYSFILE SYSLST=L.INV.COUP
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSOR ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
/FILE ADAyyyyy.SORT ,LINK=DDSORTR1
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADAINV,DB=yyyyy,IDTNAME=ADABAS5B
ADAINV COUPLE FILE=1,3,DESCRIPTOR= AA,AA
ADAINV TEMPSIZE=100,SORTSIZE=50
/LOGOFF NOSPOOL
```

Invert File

In SDF Format:

```
/.ADAINV LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A I N V INVERT FIELD (REFLECTIVE)
/REMARK *
/ASS-SYSLST L.INV.INVE
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
/SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAINV,DB=yyyyy,IDTNAME=ADABAS5B
ADAINV INVERT FILE=1
ADAINV TEMPSIZE=100,SORTSIZE=50
ADAINV FIELD= AC
ADAINV SUPDE= S1,UQ=AA(1,3),AD(2,4)
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAINV LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A I N V INVERT FIELD (REFLECTIVE)
/REMARK *
/SYSFILE SYSLST=L.INV.INVE
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSOR ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
/FILE ADAyyyyy.SORT ,LINK=DDSORTR1
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADAINV,DB=yyyyy,IDTNAME=ADABAS5B
ADAINV INVERT FILE=1
ADAINV TEMPSIZE=100,SORTSIZE=50
ADAINV FIELD= AC
ADAINV SUPDE= S1,UQ=AA(1,3),AD(2,4)
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Intermediate storage	DDTEMPR1	disk	
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	When using large files, the Sort area should be split across two volumes.*
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the recovery log option
ADARUN parameters	DDCARD	reader	Operations Manual
ADAINV parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	Messages and Codes
ADAINV messages	DDDRUCK	printer	Messages and Codes

***** Performance can be improved when sorting large files if the sort dataset is split across two volumes, but this is difficult to accomplish under OS. Two sort datasets may be specified instead. They must both be on the same device type (SORTDEV parameter), and each must be exactly half the size specified with the SORTSIZE parameter.

ADAINV JCL Example (OS/390 or z/OS)

Couple Files

Refer to ADAINVCO in the MVSJOBS dataset for this example.

```
//ADAINVCO  JOB
//*
//*    ADAINV :  COUPLE FILES
//*
//INV       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <===== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <===== DATA
```

```
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <===== WORK
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1 <===== TEMP
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1 <===== SORT
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAINV,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAINV COUPLE FILE=2,3,DESCRIPTOR='BB,BB'
ADAINV        TEMPSIZE=100,SORTSIZE=100
/*
//
```

Invert File

Refer to ADAINV in the MVSJOBS dataset for this example.

```
//ADAINVDE  JOB
//*
//*     ADAINV :  INVERT A FIELD TO A DE
//*
//INV       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <===== ASSO
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1 <===== TEMP
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1 <===== SORT
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAINV,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAINV INVERT FILE=1
ADAINV FIELD='AC'
ADAINV SUPDE='S1,UQ=AA(1,3),AD(2,4)'
ADAINV TEMPSIZE=100,SORTSIZE=100
/*
//
```

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Intermediate storage	DDTEMPR1	disk	
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	When using large files, the Sort area should be split across two volumes.*
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the recovery log option
ADARUN parameters	DDCARD	disk/terminal/ reader	Operations Manual
ADAINV parameters	DDKARTE	disk/terminal/ reader	
ADARUN messages	DDPRINT	disk/terminal/ printer	Messages and Codes
ADAINV messages	DDDRUCK	disk/terminal/ printer	Messages and Codes

***** Performance can be improved when sorting large files if the sort dataset is split across two volumes, but this is difficult to accomplish under CMS. Two sort datasets may be specified instead. They must both be on the same device type (SORTDEV parameter), and each must be exactly half the size specified with the SORTSIZE parameter.

ADAINV JCL Examples (VM/ESA or z/VM)

Couple Files

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDPRINT,DSN=ADAINV.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAINV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNINV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAINV.CONTROL,MODE=A
ADARUN
```

Contents of RUNINV CONTROL A1:

```
ADARUN PROG=ADAINV,DEVICE=dddd,DB=yyyyy
```

Contents of ADAINV CONTROL A1:

```
ADAINV COUPLE FILE=1,3,DESCRIPTOR='AA,AA'
ADAINV        TEMPSIZE=100,SORTSIZE=50
*
```

Invert File

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDPRINT,DSN=ADAINV.DDPRINT,MODE=A
DATADEF DUMP,DUMMY

DATADEF DDDRUCK,DSN=ADAINV.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNINV.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAINV.CONTROL,MODE=A
ADARUN
```

Contents of RUNINV CONTROL A1:

```
ADARUN PROG=ADAINV,DEVICE=dddd,DB=yyyyy
```

Contents of ADAINV CONTROL A1:

```
ADAINV INVERT FILE=1
ADAINV        TEMPSIZE=100,SORTSIZE=50
*
ADAINV        FIELD='AC'
ADAINV        SUPDE='S1,UQ=AA(1,3),AD(2,4)'
```

VSE/ESA

File	File Name	Storage	Logical Unit	More Information
Associator	ASSORn	disk	*	
Intermediate storage	TEMPR1	disk	*	
Sort area	SORTR1	disk	*	
Recovery log (RLOG)	RLOGR1	disk	*	Required with recovery log (RLOG) option
ADARUN parameters	– CARD CARD	reader tape disk	SYSRDR SYS000 *	
ADAINV parameters	–	reader	SYSIPT	
ADARUN messages	–	printer	SYSLST	Messages and Codes
ADAINV messages	–	printer	SYS009	Messages and Codes

***** Any programmer logical unit can be used.

ADAINV JCS Examples (VSE/ESA)

See appendix B for a description of the VSE/ESA procedures (PROCs).

Couple Files

Refer to member ADAINVCO.X for this example.

```
* $$ JOB JNM=ADAINVCO,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAINVCO
*       COUPLE FILES
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAINV,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAINV COUPLE FILE=2,3,DESCRIPTOR='BB,BB'
ADAINV        TEMPSIZE=100,SORTSIZE=100
/*
/&
* $$ EOJ
```

Invert File

Refer to member ADAINV.X for this example.

```
* $$ JOB JNM=ADAINV,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAINV
*       INVERT A FIELD TO A DESCRIPTOR
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAINV,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAINV INVERT FILE=1
ADAINV FIELD='AC'
ADAINV SUPDE='S1,UQ=AA(1,3),AD(2,4)'
ADAINV TEMPSIZE=100,SORTSIZE=100
/*
/&
* $$ EOJ
```

11

ADALOD : LOADER

Functional Overview

The ADALOD LOAD function (see page 314) loads a file into the database. Compressed records produced by the ADACMP or ADAULD utility may be used as input.

ADALOD loads each compressed record into Data Storage, builds the address converter for the file, and enters the field definitions for the file into the field definition table (FDT). ADALOD also extracts the values for all descriptors in the file together with the ISNs of all records in which the value is present, to an intermediate dataset. This dataset is then sorted into value/ISN sequence and then entered into the Associator inverted lists.

The ADALOD UPDATE function (see page 341) is used to add or delete a large number of records to/from an Adabas file. The UPDATE function requires considerably less processing time than the repetitive execution of the Adabas add/delete record commands. Records to be added may be the compressed records produced by the ADACMP or ADAULD utility. The ISNs of records to be deleted can be provided either in an input dataset or by using control statements.

Records may be added and other records deleted during a single execution of ADALOD.

LOAD : Load a File

```
ADALOD LOAD     FILE=file-number [,file-type]
                DSSIZE=size
                MAXISN=max-number-of-records
                SORTSIZE=size
                TEMPSIZE=size
                [ACRABN=starting-rabn]
                [ADAMFILE ]
                     ADAMDE={ field|ISN }
                     [ADAMOFLOW=size]
                     [ADAMPARM={ number|0 } ]
                [ALLOCATION={FORCE | NOFORCE}]
                [ ANCHOR=file-number
                     MINISN=lowest-allocated-isn,
                     NOACEXTENSION ]
                [ASSOPFAC={ padding-factor|10 } ]
                [ASSOVOLUME='Associator-extent-volume']
                [DATAFRM={ YES|NO } ]
                [DATAPFAC={ padding-factor|10 } ]
                [DATAVOLUME='Data-Storage-extent-volume']
                [DSDEV=device-type]
                [DSRABN=start-rabn]
                [DSREUSE={ YES|NO } ]
                [ETID=owner-id]
                [IGNFDT]
                [INDEXCOMPRESSION={ YES|NO } ]
                [ISNREUSE={ YES|NO } ]
                [ISNSIZE={ 3|4 } ]
                [LIP={ isn-pool-size|2000 } ]
                [LOWNERID={ owner-id-length | 0 } ]
                [LWP={ work-pool-size|1048576 } ]
                [MAXDS={ max-DS-secondary-allocation|no-limit } ]
                [MAXNI={ max-NI-secondary-allocation|no-limit } ]
```

[**MAXRECL=**{ max-compressed-record-length | max-possible }]
[**MAXUI=**{ max-UI-secondary-allocation|no-limit }]
[**MINISN=**{ lowest-allocated-isn|**1** }]
[**MIXDSDEV**]
[**NAME=**{ name|**TESTFILE** }]
[**NIRABN=s**tart-rabn]
[**NISIZE=**size]
[**NOACEXTENSION**]
[**NOUSERABEND**]
[**NUMREC=**{ max-number-of-records-to-load|all-records }]
[**PGMREFRESH=**{ **YES**|**NO** }]
[**RESTART**]
[**SKIPREC=**{ number|**0** }]
[**SORTDEV=**{ device-type|ADARUN-device }]
[**TEMPDEV=**{ device-type|ADARUN-device }]
[**TEST**]
[**UIRABN=s**tart-rabn]
[**UISIZE=**size]
[**UQDE=**descriptor-list]
[**USERISN=**{ **YES**|**NO** }]
[**VERSION=**{ **4**|**5** | **6** | **7**}]

Essential Parameters

DSSIZE : Extent Size for Data Storage

DSSIZE is the count of blocks or cylinders to be assigned to the file's Data Storage logical extent. This value must be specified. Block values must be followed by "B" (for example, 5000B).

The number can be taken directly from the Space Requirements report produced by the ADACMP utility. If the specified extent size exceeds the largest free size, ADALOD allocates as many file extents as necessary (up to a total of 5) to satisfy the request.

If a small number of records is being loaded now and a larger number of records is to be added later, the ADACMP report value should be increased in proportion to the total records to be added; otherwise, the space allocation for Data Storage (the original and four additional extents) may not be large enough to accommodate the records to be added. The file must then be unloaded and reloaded (or reordered) to increase the Data Storage space allocation. For more information, see the section **LOAD File Space Allocation** on page 334.

FILE : File Number, File Type

FILE specifies the Adabas file number and file type to be assigned to the file.

The number specified must not be currently assigned to another file in the database, unless that file was first deleted using the KEEPFDT parameter (see ADADBS DELETE function). The number must not be greater than the maximum file number defined for the database; for a checkpoint, security, or system file, the number must be 255 or lower (a trigger file can have a two-byte file number). File numbers may be assigned in any sequence.

The file type is used to indicate that the file is an Adabas system file. One of the following keywords may be specified:

CHECKPOINT	Adabas checkpoint file
SECURITY	Adabas security file
SYSFILE	Adabas system file
TRIGGER	Adabas trigger file

Notes:

1. *An existing checkpoint system file created using the ADADEF utility cannot be overwritten.*
2. *The security system file is required if Adabas Security is to be used.*
3. *In an Adabas Transaction Manager (ATM) database, SYSFILE numbers 5 and 6 are reserved for the ATM nucleus. For Adabas version 7.1, these file numbers cannot be changed. The file numbers will become more flexible in subsequent versions of Adabas.*

   Use the following parameters to load the ATM system files on an ATM database (ADARUN DTP=TM):

   ADALOD LOAD FILE=5,SYSFILE
   ADALOD LOAD FILE=6,SYSFILE

4. *If CHECKPOINT, SECURITY, or TRIGGER is specified, the contents of DD/EBAND are ignored.*
5. *CHECKPOINT, SECURITY, or SYSFILE files can be deleted only by the ADADBS DELETE function running as the only Adabas user; deleting a system file terminates Adabas when deletion is completed.*
6. *Adabas allows a maximum of eight (8) system files.*

MAXISN : Highest ISN to Be Allocated

The MAXISN parameter is required. It specifies the highest ISN that may be assigned in the file. The highest MAXISN value that Adabas permits is 4,294,967,294. There is no default value.

Note that MAXISN does **not** specify the maximum number of records that can be loaded into the file. The maximum number of records that Adabas permits in a file depends on the ISNSIZE parameter, which specifies whether ISNs in the file are 3 bytes or 4 bytes long. If ISNSIZE=3, Adabas permits up to 16,777,215 records. If ISNSIZE=4, Adabas permits up to 4,294,967,294 records.

However, the MAXISN and MINISN parameters together limit the number of records in the file. The number of possible ISNs is given by

(MAXISN – MINISN) + 1

For example, to limit a file to 10 million records, the user can specify the following values:

MAXISN=10000000	or	MAXISN=30000000
MINISN=1		MINISN=20000001

Similarly, the following values would limit a file with 4-byte ISNs to 50 million records:

MAXISN=50000000	or	MAXISN=50500000
MINISN=1		MINISN=500001

ADALOD uses the MAXISN and MINISN values when it allocates space for the address converter. Depending on the size of RABNs in the database (which is determined by the ADADEF parameter RABNSIZE), each ISN requires 3 or 4 bytes in the address converter. ADALOD multiplies the number of possible ISNs by 3 or 4 and then calculates the number of blocks that must be allocated.

If more than **(MAXISN – MINISN) + 1** records are to be loaded, and if NOACEXTENSION is not specified, ADALOD increases the MAXISN value and allocates an additional address converter extent.

SORTSIZE : Sort Size

SORTSIZE specifies the space available for the sort dataset or datasets R1/2 (SORTR2 is not supported under VSE). The value can be either cylinders (a numeric value only) or blocks (a numeric value followed by "B"). If blocks are specified, they should be equivalent to a full number of cylinders. The SORTSIZE parameter must be specified. Refer to the *Adabas DBA Reference Manual* for more information on estimating the sort space.

TEMPSIZE : Temporary Storage Size

TEMPSIZE specifies the size of the temp dataset for the file. The Temp size equals the total of TEMP space required for each descriptor in the file; see the section **LOAD File Space Allocation** on page 334 for more information. The size can be either in cylinders or blocks (followed by "B").

Optional Parameters and Subparameters

ACRABN / DSRABN / NIRABN / UIRABN : Starting RABN

Causes space allocation for the address converter (ACRABN), Data Storage (DSRABN), the normal index (NIRABN), or the upper index (UIRABN) to begin at the specified RABN.

ADAMFILE : File to Be Loaded with ADAM Option

ADAMFILE specifies the file is to be loaded using the ADAM option.

If this parameter is specified, the Data Storage RABN for each input record is calculated using a randomizing algorithm, the result of which is based on the value of the ADAM descriptor in each record. See the ADAMER utility description for additional information about using the ADAM option. If ADAMFILE is specified, ADAMDE must also be specified.

ADAMDE : ADAM Key

ADAMDE specifies the field to be used as the ADAM key.

The ADAM descriptor must be defined in the field definition table (FDT). The descriptor must have been defined with the UQ option, and **cannot**

- be a sub-, super-, hyper-, collation, or phonetic descriptor;
- be a multiple-value field;
- be a field within a periodic group;
- be variable length;
- specify the null suppression (NU) option.

If the ISN of the record is to be used as the ADAM key, ADAMDE=ISN must be specified.

This parameter must be specified when the ADAM option has been selected for the file being loaded with the ADAMFILE parameter.

ADAMOFLOW : Overflow Area Size for ADAM File

ADAMOFLOW is the size of the Data Storage area to be used for ADAM file overflow. The ADAMOFLOW value applies only if the ADAM option has been selected for the file being loaded (see ADAMFILE parameter).

ADALOD will choose a prime number which is less than DSSIZE minus ADAMOFLOW (in blocks). This prime number is used to compute the Data Storage RABN for each record. If a record does not fit into the block with the computed RABN, it is written to the next free RABN in the overflow area.

ADAMPARM : Bit Truncation for ADAM File

ADAMPARM specifies the number of bits to be truncated from the ADAM descriptor value before it is used as input to the ADAM randomizing algorithm. A value in the range 1–255 may be specified. If this parameter is omitted, a value of 0 bits (no truncation) will be used.

This parameter achieves a type of record “clustering” with nearly equal ADAM keys. ADAMPARM can be specified only when the ADAMFILE parameter has also been specified.

ALLOCATION : Action to Follow File Extent Allocation Failure

ALLOCATION specifies the action to be taken if file extent allocations cannot be obtained according to the placement parameters ACRABN, DSRABN, NIRABN, or UIRABN.

By default (that is, ALLOCATION=FORCE), the utility terminates with error if any file extent allocation cannot be met according to RABN placement parameters.

If ALLOCATION=NOFORCE is specified and any allocation with placement parameters fails, the utility retries the allocation without the placement parameter.

If insufficient space can be obtained according to the placement parameters DSRABN, NIRABN, or UIRABN, only the first extent will be made there and the rest (until the fifth extent) will be made elsewhere. But if the placement parameter ACRABN is used with ALLOCATION=FORCE, the complete space has to be available there; otherwise, the utility terminates with an error.

ANCHOR : Expanded Component / Anchor File

ANCHOR defines the base (anchor) file for either an existing or a new expanded file. If the file defined by ANCHOR is the same as that defined by the FILE parameter, the loaded file becomes the physical base (anchor) file for a new expanded logical file. Otherwise, the FILE file becomes a new component of the expanded file defined by ANCHOR.

If ANCHOR specifies a file that is not part of an expanded file, the LOAD operation defines this file and the file specified by the FILE parameter as a new expanded file. It also sets the NOACEXTENSION indicator for the file specified by ANCHOR.

If ANCHOR specifies the anchor file of an already existing expanded file, the LOAD operation adds the file specified by FILE to the expanded file.

Note:
When loading a new file to an existing expanded file, you must have exclusive update use of the anchor file as well as the file being added. This can be achieved by locking the anchor file for utility use.

Both the file specified by ANCHOR and the file specified by FILE must have the same field definition table (FDT) structure. The maximum record length (MAXRECL parameter) and any file security definitions must also be the same.

If ANCHOR is specified, the MINISN and NOACEXTENSION parameters must also be specified. Coupled files or multiclient files cannot be part of expanded files.

ASSOPFAC : Associator Padding Factor

ASSOPFAC defines the padding factor to be used for each Associator block. If not specified, the default padding factor is 10.

The value specified represents the percentage of each Associator block (padding area) that is not to be used during the loading process. The padding area is reserved for use when additional entries must be added to the block for new descriptor values or new ISNs for existing values, thereby avoiding the overhead caused by relocating overflow entries into another block.

A value in the range 1–90 may be specified. The number of bytes contained in an Associator block, minus the number of bytes reserved for padding, must be larger than the largest descriptor value contained in the file, plus 10 bytes.

A small padding factor (1–10) should be specified if little or no descriptor updating is planned. A larger padding factor (10–50) should be specified if a large amount of updating including addition of new descriptor values (or new ISNs) is planned.

ASSOVOLUME : Associator Extent Volume

Note:
The value for ASSOVOLUME must be enclosed in apostrophes.

ASSOVOLUME specifies the volume on which the file's Associator space (that is, the AC, NI, and UI extents) is to be allocated. If the requested number of blocks cannot be found on the specified volume, ADALOD retries the allocation while disregarding the ASSOVOLUME parameter.

Note:
If there are five or more blocks of unused ASSO space on the specified volume, ADALOD allocates these blocks; if this is not enough space, it ends with ERROR–060. If there are no free blocks remaining on the specified volume, ADALOD tries to allocate space on another volume.

If ACRABN, UIRABN, or NIRABN is specified, ADALOD ignores the ASSOVOLUME value when allocating the corresponding extent type. If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADALOD's default allocation rules.

DATAFRM : Overwrite ADAM Data Storage

DATAFRM controls overwriting of an ADAM file's Data Storage during loading. DATAFRM=YES (the default) forces ADALOD to reformat the Data Storage area when the file is loaded; DATAFRM=NO prevents reformatting, and is recommended when loading relatively few records because the load operation may run significantly faster. Specifying NO, however, assumes that the Data Storage area was previously formatted with the ADAFRM utility specifying FROMRABN.

Caution:
Specify DATAFRM=NO with care. If the primary Data Storage area was inxcorrectly formatted, later file processing could cause errors and unpredictable results.

DATAPFAC : Data Storage Padding Factor

DATAPFAC is the padding factor to be used for each Data Storage physical block. A percentage value in the range 1–90 may be specified. If not specified here, the default padding factor is 10.

A small padding factor (1–10) should be specified if little or no record expansion is expected. A larger padding factor (10–50) should be specified if a large amount of updating is planned that will expand the logical records.

The percentage value specified represents the portion of each Data Storage block (padding area) to be reserved during the loading process for later record expansion. The padding area is used when any given logical record within the block requires additional space as the result of record updating, thereby avoiding the overhead that would be needed to relocate the record to another block.

Since records loaded into a file can be different lengths, the padding factor cannot be exactly the percentage specified in each block. Adabas balances the size of the padding area for the different record lengths to the extent that at least 50 bytes remain in a block.

Example:

A blocksize is 1000 bytes; the padding factor is 10%. The space available for loading records (blocksize – padding-area) is therefore 900 bytes.

After loading some records, 800 bytes of the block have been used. The next record is 170 bytes long. This record cannot be loaded into the current block because less the 50 bytes would remain in the block after the record was loaded. Therefore, the record is loaded into the next block.

The current block remains filled to 800 bytes. The difference between 800 and 900 bytes (that is, –100 bytes) is used for balancing.

Suppose the next record had been 150 bytes instead of 170 bytes, and assume that the cumulative balancing value at that point in time is a negative number of bytes. The 150-byte record would be loaded because 50 bytes would remain in the block after the record was loaded (1000 – 950).

However, 50 bytes of the padding area would have been used (900 – 950) leaving +50 bytes for balancing.

For files loaded with the ADAM option, a new record is loaded into its calculated Data Storage block if space is available in the block (including the padding area). Records that cannot be stored in their calculated block are stored in another block (in this case, the padding area is not used).

DATAVOLUME : Data Storage Extent Volume

Note:
The value for DATAVOLUME must be enclosed in apostrophes.

DATAVOLUME specifies the volume on which the file's Data Storage space (DS extents) is to be allocated. If the number of blocks requested with DSSIZE cannot be found on the specified volume, ADALOD retries the allocation while disregarding the DATAVOLUME value.

If DSRABN is specified, DATAVOLUME is ignored for the related file. If DATAVOLUME is not specified, the Data Storage space is allocated according to ADALOD's default allocation rules.

DSDEV : Data Storage Device Type

DSDEV specifies the device type on which the file's Data Storage is to be loaded. There is no default value; if DSDEV is not specified, an arbitrary device type is used.

DSREUSE : Data Storage Reusage

DSREUSE indicates whether Data Storage space which becomes available is to be reused. The default is YES.

ETID : Multiclient File Owner ID

The ETID parameter assigns a new owner ID to all records being loaded into a multiclient file. It specifies the user ID identifying the owner of the records being loaded. The owner ID assigned to the records is taken from the user profile of the specified user ID.

The ETID parameter must be specified if the file is to be loaded as a multiclient file (see the LOWNERID parameter discussion on page 325) and the input file contains no owner IDs; that is, the input file was not unloaded from a multiclient source file.

ETID is optional if the input file was unloaded from a multiclient source file. In this case, the loaded records keep their original owner IDs.

The ETID parameter must not be specified when loading a non-multiclient file.

Note:
If the ETID parameter is used, the ADALOD utility requires an active nucleus. The nucleus will translate the ETID value into the internal owner ID value.

IGNFDT : Ignore Old FDT

When a file is deleted using the ADADBS DELETE function with the KEEPFDT parameter, the field definition table (FDT) remains in the Associator. When the file is again reloaded and IGNFDT is not specified, ADALOD compares the file's old FDT with the new one (security information is not compared). If both FDTs are identical, ADALOD loads the file and replaces the old FDT with the new FDT. If the FDTs are not identical, the old FDT is kept and the ADALOD operation ends with an error message.

Specifying the IGNFDT parameter causes ADALOD to ignore any existing (old) FDT for the file; no comparison is made. The new FDT replaces the old FDT, and ADALOD loads the file.

INDEXCOMPRESSION : Compress File Index

INDEXCOMPRESSION indicates whether the index of the file is loaded in compressed or uncompressed form. A compressed index usually requires less index space and improves the efficiency of index operations in the Adabas nucleus.

If INDEXCOMPRESSION is not specified, ADALOD obtains the default value from the sequential input file. If the input file was created using

- ADACMP, the default value is NO.

- ADAULD, the value of the file at the time of the unload is taken as the default.

ISNREUSE : ISN Reusage

ISNREUSE indicates whether or not an ISN freed as the result of deleting records may be reassigned to a new record. The default is NO.

ISNSIZE : 3- or 4-Byte ISN

ISNSIZE indicates whether ISNs in the file are 3 or 4 bytes long. The default is 3 bytes.

LIP : ISN Buffer Pool Size

LIP specifies the size of the ISN pool for containing ISNs and their assigned Data Storage RABNs. The value may be specified in bytes as a numeric value (“2048”) or in kilobytes as a value followed by “K” (“2K”). The default for LIP is 2000 bytes.

LIP can be used to decrease the number of address converter I/Os during loading when the USERISN=YES and the user-supplied ISNs are unsorted. Optimum performance is obtained if LIP specifies a buffer size large enough to hold all ISNs to be processed.

The length of one input record is ISNSIZE + RABNSIZE + 1. Thus the entry length is at least 7 bytes (the ISNSIZE of the file is 3 and the RABNSIZE of the database is 3) and at most 9 bytes (the ISNSIZE is 4 and the RABNSIZE is 4).

LOWNERID : Internal Owner ID Length for Multiclient File

The LOWNERID parameter specifies the length of the internal owner ID values assigned to each record for multiclient files. Valid length values are 0–8. If the LOWNERID parameter is not specified, its default value is the length of the owner IDs in the input file.

The specified or default value of the LOWNERID parameter determine whether a file is to be loaded as a multiclient or a non-multiclient file. If the effective LOWNERID value is zero, the file is loaded as a normal, non-multiclient file; if it is nonzero, the file is loaded as a multiclient file.

In combination with the ETID parameter, the LOWNERID parameter can be used to

- reload a non-multiclient file as a multiclient file;

- increase/decrease the length of the owner ID for the file; or
- remove the owner ID from the records of a file.

The following table shows the possible combinations of the LOWNERID parameter and the owner ID length in the input file.

LOWNERID Parameter Setting	Owner ID Length Value in Input File	
	0	2
0	Keep as a non-multiclient file	Convert to a non-multiclient file
1	Set up multiclient file (ETID)	Decrease owner ID length
2	Set up multiclient file (ETID)	Keep owner ID length
3	Set up multiclient file (ETID)	Increase owner ID length
(not specified)	Keep as a non-multiclient file	Keep as a multiclient file

When loading a multiclient file (the specified or default value of LOWNERID is non-zero), the ETID parameter can be specified to assign a new owner ID to all records being loaded. If the input file already contains owner IDs and ETID is omitted, all records keep their original owner IDs.

Where the table indicates the ETID parameter in the “Owner ID Length...0” column, the ETID parameter is mandatory, as there are no owner IDs given in the input file.

LWP : Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a “K”. If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADALOD run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the sort dataset, a smaller LWP value should be specified when loading relatively small files.

The minimum work pool size depends on the sort dataset’s device type:

Sort Device	Minimum LWP Bytes	Minimum LWP Kilobytes
2000	106496	104K
2314	090112	88K

	Bytes	Kilobytes
3375	131072	128K
3380	139264	136K
3390	159744	156K

MAXDS / MAXNI / MAXUI : Maximum Secondary Allocation

Specifies the maximum number of blocks per secondary extent allocation for Data Storage (MAXDS), normal index (MAXNI), or upper index (MAXUI). The value specified must be in blocks (for example, MAXNI=8000B) and cannot be more than 65535B. If no limit is specified, no limit is assumed (the default).

MAXRECL : Maximum Compressed Record Length

MAXRECL specifies the maximum compressed record length permitted for the file. The default is the maximum length supported by the device type being used.

MINISN : Lowest ISN to Be Allocated

This parameter specifies the lowest ISN that can be assigned in the file. The default is 1.

The main purpose of MINISN is to assign the low end of the ISN range for a component file of an Adabas expanded file. MINISN is required when ANCHOR is specified for an expanded file.

Use MINISN to avoid wasting Associator space in files where all records are assigned ISNs significantly greater than 1. For example, a savings bank uses account numbers as ISN numbers, and the lowest account number is 1,000,001. Specifying MINISN = 1000001 stops Adabas from allocating address converter space for ISNs 1–999999, which would be unused. For more information, see the description of the MAXISN parameter.

MIXDSDEV : Data Storage Mixed Device Types

MIXDSDEV allows the allocation of secondary Data Storage extents on different device types, and therefore with different block lengths. If MIXDSDEV is not specified (the default), Data Storage extents for the specified file must all be on the same device type.

NAME : File Name

NAME is the name to be assigned to the file. This name appears, along with data pertaining to this file, on the Database Status Report produced by the ADAREP utility. The maximum number of characters permitted is 16. The default name assigned is TESTFILE.

If the file name contains special characters or embedded blanks, the name must be enclosed within apostrophes ('...'), which themselves must be doubled if one is included in the name; for example, 'JAN''S FILE'.

NISIZE : Normal Index Size

NISIZE specifies the number of blocks or cylinders to be assigned to the normal index. A block value must be followed by "B" (for example, 5500B).

If the specified extent size exceeds the largest free size, ADALOD allocates as many file extents as necessary (up to a total of 5) to satisfy the request.

If the NISIZE parameter is omitted:

- ADALOD determines the space allocation for the normal index based on a sampling of records taken from the input dataset. Since this calculation requires additional CPU time and I/O operations, Software AG recommends setting this parameter if the size is known so that no estimation is performed.
- and INDEXCOMPRESSION=YES is set, the index size estimation made by ADALOD does not consider the index compression as it has no knowledge of the rate of compression to be expected. ADALOD may thus allocate a larger index than necessary.

If a small number of records is being loaded and a larger number of records is to be added later, the NISIZE parameter should be set to increase the Normal Index to accommodate the total record amount. For more information, see the section **LOAD File Space Allocation** on page 334.

NOACEXTENSION : Limit Address Converter Extents

If NOACEXTENSION is specified, the MAXISN defined for this file cannot be increased in the future. No additional address converter (AC) extents will be created. NOACEXTENSION applies mainly to component files comprising Adabas expanded files; if ANCHOR is specified, NOACEXTENSION must also be specified.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message "utility TERMINATED DUE TO ERROR CONDITION" is displayed and the utility terminates with condition code 20.

NUMREC : Limit Number of Records to Be Loaded

NUMREC specifies the limit on the number of records to be loaded. If NUMREC is specified, ADALOD stops after processing the specified number of records (unless an end-of-file condition on the input dataset ends ADALOD operation before that time). This option is most often used to create a subset of a file for test purposes. If this parameter is omitted, all input records are processed.

If the input dataset contains more records than specified by NUMREC, ADALOD processes the number of records specified by NUMREC and then ends with condition code 4.

PGMREFRESH : Program-Generated File Refresh

PGMREFRESH specifies whether a user program is allowed to perform a refresh operation on the file being loaded. If PGMREFRESH is specified, a refresh can be made using an E1 command, or an equivalent call to the nucleus.

RESTART : Restart Interrupted ADALOD Execution

RESTART forces an interrupted ADALOD run to be restarted, beginning with the last "restart point" reached before the interruption. The "restart point" is the latest point of execution that can be restored from the temp dataset.

If ADALOD is interrupted by a defined error condition, ADALOD issues a message indicating whether or not a restart is possible.

When restarting the ADALOD operation, the following parameters may be changed:

- TEMPSIZE can be increased to make the temp dataset larger. Note, however, that the temp dataset content contains information necessary for the restart operation, and therefore **must not be changed**;
- The SORTSIZE and SORTDEV parameters and the sort dataset can be changed.

No other parameters can be changed. The DDEBAND/EBAND and DDFILEA/FILEA datasets must remain the same.

SKIPREC : Number of Records to Be Skipped

SKIPREC specifies the number of input records to be skipped before beginning load processing. The default is 0 (no records are skipped).

SORTDEV : Sort Device Type

ADALOD uses the sort dataset to sort descriptor values. The SORTDEV parameter indicates the device type to be used for this dataset. This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

TEMPDEV : Temporary Storage Device Type

ADALOD uses the temp dataset to store intermediate data. The TEMPDEV parameter indicates the device type to be used for this dataset. This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

UISIZE : Upper Index Size

UISIZE specifies the number of blocks or cylinders to be assigned to the upper index. A block value must be followed by “B” (for example, 5500B).

If the specified extent size exceeds the largest free size, ADALOD allocates as many file extents as necessary (up to a total of 5) to satisfy the request.

If the UISIZE parameter is omitted:

- ADALOD determines the space allocation for the upper index based on a sampling of records taken from the input dataset. Since this calculation requires additional CPU time and I/O operations, Software AG recommends setting this parameter if the size is known so that no estimation is performed.
- and INDEXCOMPRESSION=YES is set, the index size estimation made by ADALOD does not consider the index compression as it has no knowledge of the rate of compression to be expected. ADALOD may thus allocate a larger index than necessary.

If a small number of records is being loaded and a larger number of records are to be added later, the UISIZE parameter should be set to increase the upper index to accommodate the total record amount. For more information, see the section **LOAD File Space Allocation** on page 334.

UQDE : Unique Descriptors

UQDE defines one or more descriptors as unique. Each descriptor specified must contain a different value in each input record. If a non-unique value is detected during ADALOD processing, ADALOD terminates with an error message.

If the unique descriptor (UQ) option was specified with the ADACMP utility, the UQDE parameter here is not required.

Adabas prevents a descriptor defined with the unique descriptor (UQ) option from being updated with an add or update command if the update would cause a duplicate value for the descriptor.

Note:
For Adabas expanded files, ADALOD can only detect unique descriptor violations within the component file. If an identical value exists for a unique descriptor in one of the other component files, ADALOD cannot detect it. You must therefore ensure that unique descriptor values remain unique throughout an expanded file.

USERISN : User ISN Assignment

USERISN=YES indicates that the USERISN option for the loaded file is to be in effect, and that the ISN for each new record is being supplied by the user in the input data. If USERISN=NO, Adabas assigns the ISN for each new record.

If USERISN is not specified, a default setting is assumed that depends on the input file itself. If the input file was created by ADACMP with the USERISN option or by ADAULD from a file having the USERISN option, the default for ADALOD operation is USERISN=YES; otherwise, the default is USERISN=NO. Specifying USERISN here overrides the existing default value.

Note:
Adabas 5.2 files initially loaded with the USERISN option do not require USERISN=YES to again be specified when the files are reloaded; ADALOD assumes the default as described above. However, Adabas 5.1 files initially loaded with the USERISN option ***must*** *have USERISN=YES specified whenever they are reloaded.*

VERSION : Input Data Format

Originally, this parameter specified the Adabas version of the output (ADACMP) datasets to be loaded into Adabas.

Because ADALOD determines the version of the sequential input dataset itself, this parameter is ignored. It is available only for compatibility with old ADALOD jobs.

Examples

Example 1;

```
ADALOD  LOAD FILE=6,MAXISN=20000,DSSIZE=20,ASSOPFAC=15,
ADALOD       DATAPFAC=15,TEMPSIZE=20,SORTSIZE=10
```

File 6 is to be loaded. The number of records initially permitted for the file is 20,000. 20 cylinders are to be allocated for Data Storage. The Associator and Data Storage block padding factors are both 15 percent. The temp and sort datasets are 20 and 10 cylinders, respectively.

Example 2:

```
ADALOD  LOAD FILE=7,MAXISN=350000,ASSOPFAC=5,MINISN=100001
ADALOD       DATAPFAC=15,DSSIZE=100,USERISN=YES
ADALOD       TEMPSIZE=200,SORTSIZE=100
```

File 7 is to be loaded. The number of records initially allocated for the file is 250,000, and the minimum is 100,001. The Associator padding factor is 5 percent. The Data Storage padding factor is 15 percent. 100 cylinders are to be allocated for Data Storage. ISNs are contained in the input. The temp and sort datasets are equal to 200 and 100 cylinders, respectively.

Example 3:

```
ADALOD  LOAD FILE=8,ADAMFILE,ADAMDE='AK'
ADALOD       ADAMPARM=4,ADAMOFLOW=5,UQDE='AK',MINISN=1
ADALOD       MAXISN=10000,DSSIZE=20,ASSOPFAC=5,DATAPFAC=5
ADALOD       TEMPSIZE=10,SORTSIZE=5
```

File 8 is to be loaded as an ADAM file. Field AK is the ADAM key. 4 bits are to be truncated from each value of AK before using the value to calculate the Data Storage RABN for the record. The size of the ADAM overflow area is 5 cylinders. The field AK is defined as a unique descriptor. The maximum number of records initially allocated for the file is 10,000. 20 cylinders are to be allocated to Data Storage, from which the five ADAM overflow cylinders are taken. The padding factor for both the Associator and Data Storage is 5 percent. The sizes of the temp and sort datasets are 10 and 5 cylinders, respectively.

Example 4:

```
ADALOD  LOAD FILE=9,NAME=INVENTORY,MAXISN=5000
ADALOD       DSSIZE=2000B,DSRABN=30629,NISIZE=300B,UISIZE=50B
ADALOD       MAXDS=1000B,MAXNI=50B,MAXUI=1B
ADALOD       INDEXCOMPRESSION=YES
ADALOD       ASSOPFAC=20,DATAPFAC=10
ADALOD       TEMPSIZE=10,SORTSIZE=5,UQDE='U1,U2'
```

File 9 is to be loaded. The text name for the file is INVENTORY. The initial space allocation for the file is for 5,000 records. 2,000 blocks are to be allocated for Data Storage, beginning with RABN 30,629. 300 blocks are to be allocated for the normal index. 50 blocks are to be allocated to the upper index. The maximum allocations per secondary extent for Data Storage, normal index and upper index are 1000 blocks, 50 blocks, and 1 block respectively. The index is to be compressed. The padding factor for the Associator is 20 percent. The padding factor for Data Storage is 10 percent. The sizes of the temp and sort datasets are 10 and 5 cylinders respectively. Descriptors U1 and U2 are defined as unique descriptors.

Example 5:

```
ADALOD  LOAD FILE=2,SECURITY
ADALOD       DSSIZE=20B,MAXISN=2000,NISIZE=20B,UISIZE=5B
ADALOD       TEMPSIZE=10,SORTSIZE=5
```

File 2 is to be loaded as an Adabas security file. The DDEBAND contents are ignored. Space is allocated for Data Storage (20 blocks), for the address converter (2000 ISNs), the normal index (20 blocks), and the upper index (5 blocks). The temp size is 10 cylinders, and the sort area size is 5 cylinders.

LOAD Data and Space Requirements

The following general information describes data requirements for LOAD operation, and how ADALOD LOAD allocates space. For more information about space allocation, refer to the *Adabas DBA Reference Manual*.

Input Data for LOAD Operations

Compressed data records produced by the ADACMP or ADAULD utility may be used as input to ADALOD. If output from an ADAULD utility run made with the MODE=SHORT option is used as ADALOD input, any descriptor information will be removed from the FDT, and no index will exist for the file.

LOAD File Space Allocation

ADALOD allocates space for the normal index (NI), upper index (UI), address converter (AC), Data Storage, and the temp area for the file being loaded.

Index Space Allocation

If the NISIZE and/or the UISIZE parameters are supplied, allocation is made using the user-supplied values. If these parameters are not supplied, ADALOD allocates space for these indexes based on a sampling of the values present for each descriptor.

Descriptor values are sampled as follows:

1. ADALOD reads the compressed input, stores the records into Data Storage, extracts each value for each descriptor and writes these values to the temp dataset. Each temp block contains values for one descriptor only. At the end of this processing phase, the following information is present:
   - number of values for each descriptor
   - number of bytes required for each descriptor
   - temp RABNs used for each descriptor

   For unique descriptors, the NI space requirement is equal to the temp size used. For non-unique descriptors, the number of duplicate values must be determined. Each duplicate value's space requirement must be estimated and then subtracted from the number of bytes required. The result is the NI size required for the duplicate descriptor.

The number of duplicate values is determined by reading up to 16 temp blocks containing values for a single descriptor. These values are sorted to determine how many are duplicates. The resulting count of duplicate values is multiplied by the factor:

$$\frac{\text{total number of values}}{\text{number of values in the sample}}$$

The result is the estimated number of identical descriptor values present in the entire file for this descriptor. This space requirement is subtracted from the temp size estimate.

2. The upper index (UI) size is computed after all normal index (NI) and temp sizes are available.

3. The NI and UI sizes are each multiplied by the result of:

$$\frac{\textbf{(MAXISN – MINISN) + 1}}{\text{number of records being loaded}}$$

For example, if 10000 records require 10 blocks of UI space and 500 blocks of NI space with MINISN = 1 (the default), the specification of MAXISN = 60000 causes 60 UI blocks and 3000 NI blocks to be allocated:

$$10 \cdot \frac{60000}{10000} = 60 \text{ UI Blocks}$$

$$500 \cdot \frac{60000}{10000} = 3000 \text{ NI Blocks}$$

However, this calculation is not made if USERISN=YES is in effect.

By setting MAXISN appropriately, it is therefore possible to increase the size allocation for files in which a small number of records are being loaded and for which a much larger number of records are to be added subsequently.

If the NISIZE and UISIZE parameters have been specified, the space allocation is made using unassigned Associator RABNs. If the NIRABN and/or the UIRABN parameters are supplied, space allocation is made at the user-specified RABN.

Address Converter Space Allocation

The address converter allocation is based on the MAXISN and MINISN values for the file. ADALOD allocates the blocks needed to contain the number of bytes calculated by the formula **RABNSIZE • ((MAXISN – MINISN) + 1)**. If the ACRABN parameter has been specified, ADALOD allocates the address converter beginning with the user-specified block number; otherwise, it uses unassigned Associator RABNs.

Data Storage Space Allocation

Data Storage allocation is based upon the value specified with the DSSIZE parameter. If the DSRABN parameter has been specified, the allocation is made beginning with the user-specified block number; otherwise, unassigned Data Storage RABNs are used.

If there are different device types in the database, Data Storage allocation can be forced on a specified device type by specifying DSDEV. The MIXDSDEV parameter permits Data Storage allocation on different device types, assuming the device types can store records with the length specified by MAXRECL.

Temp Area Space Allocation

For each descriptor, ADALOD generates a list of the values and ISNs of the records containing the value, and writes this information to the temp dataset. The space required for descriptor information is equal to the sum of the space required for each descriptor. The space needed for each descriptor can be calculated using the following formula:

SP = N • NPE • NMU • (L + 4)

—where

SP	is the space required for the descriptor (in bytes).
N	is the number of records being loaded.
NPE	is the average number of occurrences, if the descriptor is contained in a periodic group. If not in a periodic group, NPE equals 1.
NMU	is the average number of occurrences, if the descriptor is a multiple-value field. If not a multiple-value field, NMU equals 1.
L	is the average length (after compression) of each value for the descriptor.

Example:

A file containing 20,000 records is being loaded. The file contains two descriptors (AA and CC). Descriptor AA has 1 value in each record and the average compressed value length is 3 bytes. Descriptor CC has an average of 10 values in each record and the average compressed value length is equal to 4 bytes.

Field Definitions:

```
01,AA,5,U,DE
01,CC,12,A,DE,MU
```

- Space requirement for AA.

  SP = 20,000 • 1 • (3 + 4)
  SP = 140,000 bytes

- Space requirement for CC.

  SP = 20,000 • 10 • (4 + 4)
  SP = 1,600,000 bytes

- Total space requirement = **1,740,000 bytes.**

  The number of cylinders required may be calculated by dividing the number of blocks required by the number of blocks per cylinder.

  For a model 3380 device type:

$$\text{Blocks required} = \frac{1,740,000 \text{ bytes}}{7476 \text{ bytes per block}} = 232+, \text{ or } 233 \text{ blocks}$$

$$\text{Cylinders required} = \frac{233 \text{ blocks}}{90 \text{ blocks per cylinder}} = 2+, \text{ or } 3 \text{ cylinders}$$

Associator Updating by LOAD

ADALOD then sorts the descriptor values collected in the input phase and enters the sorted values into the normal index and upper index. If the allocated index space is not enough for the normal index or upper index, ADALOD allocates up to four additional extents.

Each additional extent allocated is equal to about 25 percent of the total current space allocated to the index. If insufficient space is available for the additional extent or the maximum of five extents has already been allocated, ADALOD terminates with an error message.

Loading Expanded Files

An expanded file is made up of a series of normal Adabas physical files. The number sequence of the files within the expanded file is arbitrary. The first file may be file 53; the second, file 127; the third, 13, and so on. ISNs assigned to each component file must be unique; no two files can contain the same ISN. The ISN range over all files must be in ascending order; however, there can be gaps in the sequence.

The total number of records in an expanded-file chain cannot exceed 4,294,967,294.

The sequence of physical component files that build an expanded logical file is defined by the ANCHOR parameter, which defines the first component file (anchor) in the sequence. The anchor file is loaded just as any other Adabas file; each additional component file must be loaded with the ANCHOR parameter referring to the anchor file. ADALOD inserts the new physical file into the existing expanded file chain according to the range of ISNs assigned to the added file. Each added component file must also specify the NOACEXTENSION parameter when being loaded to prevent Adabas from assigning new ISNs to a component file.

ADALOD processes only the anchor file and the single physical (component) files that compose an expanded file, and not the complete expanded file itself.

Loading Data into an Expanded File

To load data (for example, several million records) into different physical files, the input data must first be divided into several DDEBAND/EBAND input files. The DDEBAND/EBAND file data may be mapped into the component files using the SKIPREC and NUMREC parameters; however, one-to-one mapping without skipping or limits is recommended. This avoids the need to read records that will not be used later, and thus improves performance.

Examples:

The following examples, which show parts of one or more jobs for loading an expanded file, illustrate the mapping of DDEBAND/EBAND file data into component files:

```
//DDEBAND    DD DSN=LOAD.DATA.FILE1,...
//DDKARTE    DD *
ADALOD LOAD FILE=40,NAME='XXX_Part1'
ADALOD      MINISN=1,MAXISN=10000000,NOACEXTENSION
ADALOD      NUMREC=10000000
ADALOD        DSSIZE=...,NISIZE=...,UISIZE...
ADALOD        SORTSIZE=...,TEMPSIZE=...
            .
            .

//DDEBAND    DD DSN=LOAD.DATA.FILE1,...
//DDKARTE    DD *
ADALOD LOAD FILE=41,NAME='XXX_Part2',ANCHOR=40
ADALOD      MINISN=10000001,MAXISN=20000000,NOACEXTENSION
ADALOD      NUMREC=10000000,SKIPREC=10000000
ADALOD        DSSIZE=...,NISIZE=...,UISIZE...
ADALOD        SORTSIZE=...,TEMPSIZE=...
            .
            .

//DDEBAND    DD DSN=LOAD.DATA.FILE2,...
//DDKARTE    DD *
ADALOD LOAD FILE=35,NAME='XXX_Part2',ANCHOR=40
ADALOD      MINISN=20000001,MAXISN=30000000,NOACEXTENSION
ADALOD      NUMREC=10000000
ADALOD        DSSIZE=...,NISIZE=...,UISIZE...
ADALOD        SORTSIZE=...,TEMPSIZE=...
            .
            .
```

Loading Multiclient Files

Note:
A multiclient file cannot be made part of an expanded file, and an expanded file cannot be converted to a multiclient file.

A multiclient file stores records for multiple users or groups of users. It divides the physical file into multiple logical files by attaching an owner ID to each record. Each user can access only the subset of records that is associated with the user's owner ID.

For any installed external security package such as RACF or CA-Top Secret, a user is still identified by either Natural ETID or LOGON ID. The owner ID is assigned to a user ID. A user ID can have only one owner ID, but an owner ID can belong to more than one user.

The ADALOD LOAD function uses the LOWNERID and ETID parameters to support the migration of an application from a standard to a multiclient environment. The parameters work together to define owner IDs and determine whether a file is a multiclient file.

LOWNERID specifies the length of the internal owner ID values assigned to each record for multiclient files. In combination with the ETID parameter, the LOWNERID parameter can be used to reload a standard file as a multiclient file, change the length of the owner ID for the file, or remove the owner ID from the records of a file.

If the LOWNERID parameter is not specified, the length of the owner ID for the input file (if any) remains the same.

ETID assigns a new owner ID to all records being loaded into a multiclient file, and must be specified if the input file contains no owner IDs; that is, the input file was not unloaded from a multiclient source file.

Examples of Loading/Updating Multiclient Files

```
ADALOD LOAD FILE=20,LOWNERID=2,NUMREC=0
```

Creates file 20 as a multiclient file. The length of the internal owner ID is two bytes, but no actual owner ID (ETID) is specified. No records are actually loaded in the file (NUMREC=0).

ADALOD LOAD FILE=20,LOWNERID=2,ETID=USER1

Creates file 20 as a multiclient file, load all supplied records, and assign them to user USER1. The length of the internal owner ID is two bytes.

ADALOD UPDATE FILE=20,ETID=USER2

Performs a mass update to add records to file 20, a multiclient file. Load all the new records and assign them to USER2.

UPDATE : Add/Delete Records

Warning:
If ADALOD UPDATE ends abnormally (due to insufficient space, for example), updates made to the file before the abnormal ending cannot be "backed out". Software AG therefore recommends that you perform ADASAV SAVE on the file before you run ADALOD UPDATE.

The UPDATE function adds and/or deletes a large number of records (ISNs) to and/or from an existing file. A single UPDATE operation can both add and delete ISNs.

Records to be added must be in compressed (ADACMP or ADAULD output) form and be in the DDEBAND/EBAND input dataset.

ISNs to be deleted must be specified by either or both of the DDISN and DELISN parameters.

Notes:

1. *The UPDATE function cannot be used with an Adabas system file if the Adabas nucleus is active, and cannot be used to change the checkpoint or security files.*
2. *A multiclient file cannot be made part of an expanded file, and an expanded file cannot be converted to a multiclient file.*

```
ADALOD UPDATE   FILE=file-number
                SORTSIZE=size
                TEMPSIZE=size
                [DDISN]
                [DELISN=isn-list]
                [DSREUSE={ YES|NO } ]
                [ETID=multiclient-file-owner-id]
                [ISNREUSE={ YES|NO }]
                [LIP={ isn-pool-size|2000 } ]
                [LWP={ work-pool-size|1048576 } ]
                [MAXISN=number
                     [ACRABN=starting-rabn]
                     [ASSOVOLUME='Associator-extent-volume' ] ]
                [NOUSERABEND]
                [NUMREC=number]
                [PASSWORD=password]
                [RESTART]
                [SAVEDREC]
                [SKIPREC={ number|0 } ]
                [SORTDEV={ device-type | ADARUN-device } ]
                [TEMPDEV={ device-type | ADARUN-device } ]
                [TEST]
                [USERISN={ YES|NO } ]
```

Essential Parameters

FILE : File Number

FILE specifies the number of the file to be updated. If a component file of an Adabas expanded file is specified, only that component file is updated; the other component files must be updated in separate UPDATE operations.

SORTSIZE : Sort Size

SORTSIZE is the number of blocks or cylinders available for the sort dataset.

TEMPSIZE : Temporary Storage Size

TEMPSIZE is the number of blocks or cylinders available for the temp dataset.

Optional Parameters and Subparameters

ACRABN : Starting RABN for Address Converter

ACRABN causes additional space allocation for the address converter to begin at the specified RABN. ACRABN is effective only if MAXISN specifies an increase for the file's address converter.

ASSOVOLUME : Associator Extent Volume

Note:
The value for ASSOVOLUME must be enclosed in apostrophes.

ASSOVOLUME is effective only if MAXISN specifies an increase for the file's address converter.

ASSOVOLUME specifies the volume on which the file's address converter extents is to be allocated. If the requested number of blocks cannot be found on the specified volume, ADALOD retries the allocation while disregarding the ASSOVOLUME parameter.

If ACRABN is specified, ADALOD ignores the ASSOVOLUME value when allocating the address converter extent type. If ASSOVOLUME is not specified, the file's Associator space is allocated according to ADALOD's default allocation rules.

DDISN : Read ISNs to Be Deleted from Sequential Dataset

If DDISN is specified, ISNs to be deleted are read from the DDISN/ISN sequential dataset. If both the DDISN and DELISN parameters are specified, the ISNs from the two lists are merged. The DDISN/ISN dataset must have variable or variable blocked records. See the section **Formats for Specifying ISNs** on page 349 for more information.

When the UPDATE function is executed, all ISNs are first read and stored in the ISN pool in the order they occur. The size of the ISN pool (specified by LIP) must be large enough to store all data read from DDISN/ISN.

The records are then sorted in ascending order. Overlapping ranges and duplicate ISNs are not allowed. ISNs not found during processing are ignored.

When deleting ISNs from an Adabas expanded file, you can specify the complete ISN list for all component files; the UPDATE function automatically selects only the ISNs that are appropriate for the component file being processed.

DELISN : ISNs to Be Deleted

DELISN specifies a list of the ISNs of records to be deleted. If both DDISN and DELISN are specified, the ISNs from the two lists are merged. A range list may be specified as:

DELISN=10–80,90,100–110

Overlapping ranges and duplicate ISNs are not allowed. You can specify, at most, 32 single ISNs or ISN ranges. When deleting ISNs from an Adabas expanded file, you can specify the complete list for all component files. The UPDATE function selects the appropriate ISNs from the list and deletes them from the component file.

DSREUSE : Data Storage Reusage

DSREUSE indicates whether or not Data Storage space that becomes available as a result of a record deletion is to be reused.

This parameter is in effect for the execution of the UPDATE function only. The permanent setting of DSREUSE is not changed. That permanent setting is the default if this value is not specified.

ETID : Multiclient File Owner ID

The ETID parameter assigns a new owner ID to all records being added to an existing multiclient file. The owner ID is automatically adjusted to the length for owner IDs specified by LOWNERID when the multiclient file was last loaded. If no ETID is specified, all loaded records keep their owner IDs specified on the input source.

The ETID parameter must be specified if the existing file is multiclient and the input file was not unloaded from a multiclient file. ETID must not be specified if the existing file is a non-multiclient file.

Note:
If the ETID parameter is used, the ADALOD utility requires an active nucleus. The nucleus will translate the ETID value into the internal owner ID value.

ISNREUSE : ISN Reusage

ISNREUSE indicates whether the ISN for a deleted record can be reassigned to a new record.

This ISNREUSE setting is in effect only during execution of the UPDATE function. The permanent ISNREUSE setting is unchanged. The permanent setting is the default if this value is not specified.

LIP : ISN Work Pool Size

LIP specifies the size of the work pool for containing ISNs to be deleted. Four bytes per ISN and eight bytes per ISN range are required in this pool. The value may be specified in bytes as a numeric value (“2048”) or in kilobytes as a value followed by “K” (“2K”). The default for LIP is 2000 bytes.

LWP : Work Pool Size

LWP specifies the size of the work pool to be used for descriptor value sorting. The value can be specified in bytes or kilobytes followed by a “K”. If no value is specified, the default is 1048576 bytes (or 1024K); however, to shorten ADALOD run time for files with very long descriptors or an unusually large number of descriptors, set LWP to a higher value. To avoid problems with the Sort dataset, a smaller LWP value should be specified when updating relatively small files.

The minimum work pool size depends on the sort dataset’s device type:

Sort Device	Minimum LWP Bytes	Minimum LWP Kilobytes
2000	106496	104K
2314	090112	88K
3375	131072	128K
3380	139264	136K
3390	159744	156K

MAXISN : Highest ISN to Be Allocated to the File

The MAXISN parameter may be used to specify a new setting for the file. This parameter should be used if the current record count plus the number of ISNs (records) to be added exceeds the current MAXISN setting. The specified larger value determines the additional space required for the address converter, and causes ADALOD to allocate a new extent. A smaller MAXISN value causes no change in the address converter space.

Note:
The MAXISN setting for a file cannot be increased if the file was last loaded with NOACEXTENSION active.

The MAXISN setting should be increased by an amount suitable for all planned expansion; this avoids using up the address converter extent too quickly, and alleviates the need to either unload and reload the file or run the ADAORD REORFASSO utility because the maximum of five address converter extents has been allocated.

With the optional ACRABN parameter, the beginning of the new address converter extent can be set to a specific RABN number. See the ACRABN parameter description for more information.

If the MAXISN parameter is omitted, ADALOD allocates new address converter extents only if the old MAXISN value is exceeded.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NUMREC : Limit Number of Records to Be Added

NUMREC limits the number of records to be added. If NUMREC is specified, ADALOD processing terminates after adding the number of records specified (unless an end-of-file condition on the input dataset has already caused ADALOD termination). If this parameter is omitted, **all** input records are added.

If the input dataset contains more records than specified by NUMREC, ADALOD adds the number of records specified by NUMREC and then terminates with condition code 4.

PASSWORD : File Password

If the file to be updated is password-protected, the parameter must be used to provide a valid password. There is no default for PASSWORD.

RESTART : Restart Interrupted ADALOD Execution

RESTART forces an interrupted ADALOD run to be restarted, beginning with the last “restart point” reached before the interruption. The “restart point” is the latest point of execution that can be restored from the Temp dataset.

If ADALOD is interrupted by a defined error condition, ADALOD issues a message indicating whether or not a restart is possible.

When restarting the ADALOD operation, the following parameters may be changed:

- TEMPSIZE can be increased to make the temp dataset larger. Note, however, that the temp dataset contents **must not be changed** because it contains information necessary for the restart operation;
- The SORTSIZE and SORTDEV parameters and the sort dataset can be changed.

No other parameters can be changed. The DDEBAND/EBAND, DDFILEA/FILEA and DDISN/ISN datasets must remain the same.

SAVEDREC : Save Deleted Records on a Sequential File

SAVEDREC indicates that deleted records are to be written to a sequential dataset. The format of the dataset is identical to that created by the ADAULD utility with the MODE=SHORT option.

SKIPREC : Number of Records to Be Skipped

SKIPREC is the number of input records to be skipped before beginning to process updates. The default is 0 (no records are skipped).

SORTDEV : Sort Device Type

ADALOD uses the sort dataset to sort descriptor values. The SORTDEV parameter indicates the device type to be used for this dataset. This parameter is required only if the device type to be used is different from that specified by the ADARUN DEVICE parameter.

TEMPDEV : Temporary Storage Device Type

ADALOD uses the temp dataset to store intermediate data. The TEMPDEV parameter indicates the device type to be used for this dataset. This parameter is required only if the device type to be used is different from the standard device type assigned to Temp by the ADARUN DEVICE parameter.

TEST : Test Syntax

The TEST parameter tests the operation syntax without actually performing the operation. Only the syntax of the specified parameters can be tested; not the validity of values and variables.

USERISN : User ISN Assignment

USERISN=YES indicates that the USERISN option for the file is to be in effect, and that the ISN for each new record is being supplied by the user in the input data. If USERISN=NO, Adabas assigns the ISN for each new record.

The specified USERISN setting is effective only while the UPDATE function is executing. The permanent USERISN setting is not changed, and is the default if this parameter is not specified.

When performing an ADALOD UPDATE function on a file with a hyperdescriptor for which the hyperexit changed the ISNs of descriptor values, USERISN=YES is no longer required for the add/load operation.

When adding records **from** a non-USERISN=YES file, the ADALOD parameter USERISN=NO must be specified and the file to be updated must have the USERISN option. This feature is useful for Adabas Text Retrieval (TRS).

Examples

Example 1:

```
ADALOD  UPDATE FILE=6,MAXISN=18000
ADALOD         TEMPSIZE=10,SORTSIZE=5
```

Records are to be added to file 6. The MAXISN for the file is to be increased to 18,000.

Example 2:

```
ADALOD  UPDATE FILE=7,TEMPSIZE=10,
ADALOD         ETID=USER3,SORTSIZE=5
```

Records with user’s owner ID of USER3 are to be added to multiclient file 7.

Example 3:

```
ADALOD  UPDATE FILE=8,DELISN=1000-1999,5000-5999
ADALOD         TEMPSIZE=10,SORTSIZE=5
```

The records with ISNs 1,000 to 1,999 and 5,000 to 5,999 are to be deleted from file 8. If an input dataset is provided, records are to be added.

Example 4:

```
ADALOD  UPDATE FILE=6
ADALOD         DDISN,SAVEDREC
ADALOD         TEMPSIZE=10,SORTSIZE=5
```

Records are to be deleted from file 6. The ISNs of the records to be deleted are contained in an input dataset. The deleted records are to be saved on an output dataset.

Example 5:

```
ADALOD  UPDATE FILE=6,DDISN,LIP=20K,SKIPREC=500
ADALOD         TEMPSIZE=5,SORTSIZE=10
```

Records are to be added and deleted from file 6. The ISNs which identify the records to be deleted are contained in an input dataset (DDISN). The size of the ISN pool is set to 20K. The first 500 records on the input dataset are to be skipped.

Formats for Specifying ISNs

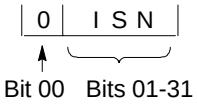
There are two formats for specifying ISNs in the DDISN or ISN dataset. The first format can be used in all cases where only 31-bit ISNs are specified. A record can contain a mix of single ISNs and ranges of ISNs.

The second format supports 32-bit ISNs and can only be used with Adabas version 6 and above. Each record can specify **either** single ISNs (indicated by X‘00000000’ in the first fullword) **or** ranges of ISNs (indicated by X‘FFFFFFFF’ in the first fullword).

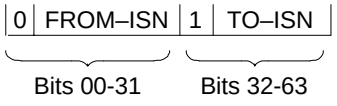
If the first fullword in a record contains a value other than X‘00000000’ or X‘FFFFFFFF’, it is assumed to be the 31-bit format. The DDISN/ISN dataset can contain records in both formats.

Format 1 : 31-Bit Format

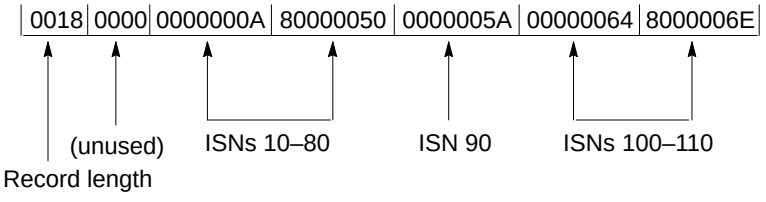
A single ISN requires 4 bytes. Set the high-order bit to 0 and specify the ISN in bits 01–31:



A range of ISNs requires 8 bytes. In the first four bytes, specify the first ISN in the range as a single ISN; in the next four bytes, set the high-order bit to 1 and specify the last ISN:



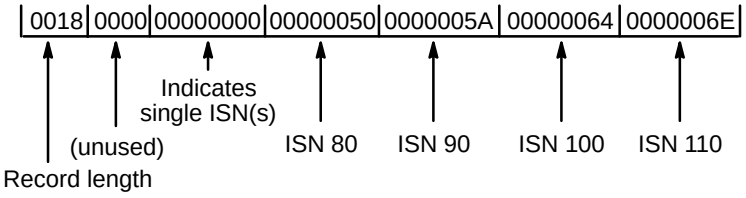
The following example shows a variable-length record containing the equivalent of DELISN=10–80,90,100–110:



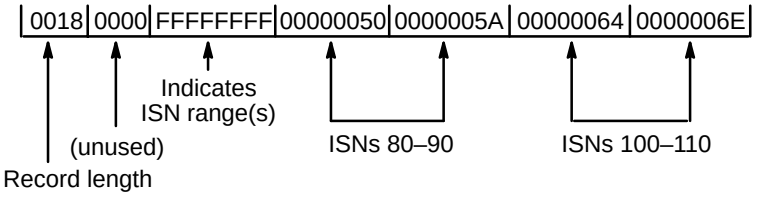
Format 2 : 32-Bit Format

In the 32-bit format, the first fullword in each record indicates whether the record contains single ISNs or ranges of ISNs. To indicate single ISNs, put zero in the first fullword (X‘00000000’); to indicate ranges of ISNs, put –1 (X‘FFFFFFFF’). In the following example, the first record contains single ISNs; the second record contains ranges. The two records are identical except for the indicator in the first fullword.

Equivalent of DELISN=80,90,100,110:



Equivalent of DELISN=80–90,100–110:



UPDATE Data and Space Requirements

The following general information describes data requirements for UPDATE operation, and how ADALOD UPDATE allocates space. For more information about space allocation, refer to the *Adabas DBA Reference Manual*.

Input Data for UPDATE Operations

Records to be added must be in the form of compressed data records produced by the ADACMP or ADAULD utility. The field definitions used for the ADACMP run must agree with the definitions for the file to which the records will be added as contained in the field definition table (FDT).

Note:
Records being added to a ciphered file must already be encrypted using the same cipher code as was used for the records already in the file.

The ISNs of records to be deleted may be provided with the DELISN parameter and/or in an input dataset. If provided in an input dataset, each ISN must be provided as a 4-byte binary number. The dataset must have the record format VARIABLE BLOCKED. If desired, all ISNs to be added to or deleted from an Adabas expanded file can be specified; the UPDATE function selects the appropriate ISNs for the component file being processed.

UPDATE Space Allocation

If records are to be added and a larger MAXISN value has been specified, an additional address converter extent will be allocated by ADALOD. The size of the new extent is based on the difference between the new MAXISN and the previous MAXISN setting. If either insufficient space is available for the new extent or the maximum of five extents has already been allocated, processing ends with an error message.

If an additional Data Storage extent is required, ADALOD allocates an additional extent equal to approximately 25 percent of the total size of the Data Storage extents currently allocated to the file. As for the address converter, processing ends with an error message if either sufficient space is not available for the added extent or the maximum of five extents has already been allocated.

Generating UPDATE Descriptor Information

When adding records, ADALOD UPDATE generates a list of all descriptor values and the corresponding ISNs of the new records, and writes this information to the temp dataset.

Associator Updating with UPDATE

Before processing the input, ADALOD UPDATE copies the file's existing normal index to the temp dataset, but removes the descriptor values of any ISNs to be deleted.

ADALOD sorts the information written to temp during the input phase and merges the sorted values with the current normal index. The normal index is reordered during this process, and the Associator block padding factor is reestablished for each block. A new upper index is then created.

Empty space in partially filled blocks resulting from descriptor updating is reused. This can increase the number of empty blocks at the end of the index. Although one or more normal index and/or upper index extents may become empty as the result of the reorder process, ADALOD does not condense, delete, or change the size of these extents.

If new free space is needed for the normal index or upper index, ADALOD allocates an additional extent (or extents). Each additional extent allocated is equal to approximately 25 percent of the total current space allocated to the index. If insufficient space is available for the additional extent or if the maximum of five extents has already been allocated, ADALOD terminates with an error message.

Mass Updates of Expanded Files

Using ADALOD UPDATE for a mass update to an expanded file, records must be added to or deleted from each component file individually. However, each component file can be processed using the same ADALOD commands.

When deleting a record with DELISN or DDISN, the complete list of ISNs to be deleted from all component files can be supplied. ADALOD automatically selects only the ISN values from the specified range that is appropriate for the component file currently being processed.

The same is true when adding new records with USERISN=YES.

When new expanded file records are being added with USERISN=NO but no free ISN is found, the loader cannot allocate a new address converter extent since the ISN range cannot be increased (NOACEXTENSION is active for all component files). Instead, ADALOD creates the index as though end-of-file had been reached. The remaining records not loaded may be added later to another component file using the SKIPREC parameter.

ADALOD does not check for unique descriptor values across component file boundaries.

Example:

The following is an example for performing a mass update to an expanded file (only the relevant parts of the complete jobs are shown):

```
               .
               .
//DDEBAND    DD DSN=MOREDATA.LOAD.PART1-2,...
//DDKARTE    DD *
ADALOD UPDATE FILE=40,USERISN=YES
ADALOD         DELISN=9000001-9500000,12000001-14000000
ADALOD         SORTSIZE=...,TEMPSIZE=...
               .
               .
//DDEBAND    DD DSN=MOREDATA.LOAD.PART1-2,...
//DDKARTE    DD *
ADALOD LOAD    FILE=41,USERISN=YES
ADALOD         DELISN=9000001-9500000,12000001-14000000
ADALOD         SORTSIZE=...,TEMPSIZE=...
               .
               -
```

Loader Storage Requirements and Use

Static Storage

Static	Type*	Size
Modules ADARUN, ADALOD	A	approximately 180 kilobytes

Dynamic Storage

Dynamic	Type*	Size
Sort work pool	A	LWP
General work pool	A	6 • (Associator block size)
I/O buffer for Associator	A	Associator block size
ISN pool	A	LIP
I/O buffer for Data Storage	A	Data Storage block size
AC bitmap	A	4K bytes
I/O buffer for temp	A	temp block size
DVT splitting	A	temp block size • number of descriptors
Internal descriptor table	A	number of descriptors • 74
I/O buffer for DDEBAND/EBAND	O	DDEBAND/EBAND block size
I/O buffer for DDOLD/OLD	O	DDOLD/OLD block size
I/O buffer for DDISN/ISN data	O	DDISN/ISN block size
I/O buffer if records must be written to temp overflow	O	DDFILEA/FILEA block size

***** Type A is always used; type O is used only if needed.

Temp Dataset Space Usage

ADALOD uses the temp dataset to store the following information:

- restart information;
- Data Storage RABN/ISN for each record to be deleted (UPDATE only);
- contents of the normal index at the start of the operation (UPDATE only);
- descriptor values obtained from the input dataset;
- ADAM overflow area (ADAM files only).

Sequential Temp Dataset

If the temp dataset is filled while collecting descriptor values from the input dataset, ADALOD temporarily writes the remaining descriptors to the sequential temp file DD/FILEA (if specified in the JCL). The descriptors are later read back in when the new index is built.

If actually called, DD/FILEA makes ADALOD operation considerably slower than specifying a temp dataset that is large enough to hold all descriptor values. The DD/FILEA TEMP dataset should normally be used only as a "safety net" to ensure adequate space for all descriptors during ADALOD operation. Specifying the DD/FILEA temp file therefore avoids an ADALOD ABEND caused by a temp area overrun.

Notes:

1. *ADALOD writes only descriptor values from the DD/EBAND input file to DD/FILEA.*
2. *The normal temp dataset must be large enough to hold all values for each single descriptor.*
3. *If the UPDATE operation accesses a file that is coupled, uncouple the files, using ADADBS or Online Services and try the UPDATE again.*

ADALOD Space/Statistics Report

During LOAD or UPDATE operation, ADALOD prints a report on the message output dataset (DDDRUCK for MVS and VM systems, SYS009 for VSE systems, or SYSOUT for BS2000). The report shows the following information:

- ADALOD function executed (LOAD or UPDATE), and the database/file affected;
- Estimated NI/UI sizes (shown for the LOAD function only if the NI/UISIZE parameters were not specified);
- Available and used file space, by Adabas component (shown for the LOAD function only);
- Current RABNs assigned for the file (shown for the LOAD function only);
- File processing statistics (records processed and system storage used).

Example of the ADALOD LOAD report:

```
PARAMETERS:

ADALOD  LOAD FILE...
         .
         .

FUNCTION TO BE EXECUTED:

LOAD FILE NUMBER    7  (MYOWNFILE     )
INTO DATABASE    0013  (MYBESTDB      )

AVAILABLE SPACE:      (LOAD function only)

        I FILE I  DEV   I  NUMBER OF  I      FROM      TO    I
        I LAY- I  TYPE  I   BLOCKS    I      RABN      RABN  I
        I OUT  I        I             I                      I
        I------I--------I-------------I----------------------I
        I ASSO I  3380  I      2695   I       137      2831  I
        I DATA I  3380  I      1339   I         3      1341  I
        ------------------------------------------------------

ESTIMATED NORMAL INDEX SIZE = 37 BLOCKS
ESTIMATED UPPER INDEX SIZE = 8 BLOCKS
```

```
TOP ISN = 773, MAX ISN EXPECTED = 1335

I FILE  I  DEV   I  LIST  I   ALLOC    I    FROM      TO  I   UNUSED   I
I LAY-  I  TYPE  I  TYPE  I   SPACE    I    RABN    RABN  I   SPACE    I
I OUT   I        I        I  (BL0CKS)  I                  I  (BLOCKS)  I
-------------------------------------------------------------------------
I ASSO  I  3380  I   AC   I         2  I     137     138 I         O  I
I ASSO  I  3380  I   UI   I         8  I     139     146 I         O  I
I ASSO  I  3380  I   NI   I        37  I     147     183 I        15  I
I DATA  I  3380  I   DS   I        60  I       3      62 I        48  I
-------------------------------------------------------------------------
```

PROCESSING STATISTICS:

```
            773 INPUT RECORDS PROCESSED
             14 BLOCKS USED ON TEMP-DATASET (0%)
              0 BLOCKS USED ON SORT PART 1  (0%)
              0 BLOCKS USED ON SORT PART 2  (0%)
          51824 BYTES OF STORAGE USED TO STORE RECORDS
```

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADALOD with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems and shows examples of each of the job streams.

Note:
When running with the optional Recovery Aid (RLOG), all temporary datasets must also be cataloged in the job control.

Collation with User Exit

If a collation user exit is to be used during ADALOD execution, the ADARUN CDXnn parameter must be specified for the utility run.

Used in conjunction with the universal encoding support (UES), the format of the collation descriptor user exit parameter is

ADARUN CDXnn=exit-name

—where

nn	is the number of the collation descriptor exit, a two-digit decimal integer in the range 01–08 inclusive.
exit-name	is the name of the user routine that gets control at the collation descriptor exit; the name can be up to 8 characters long.

Only one program may be specified for each collation descriptor exit. Up to 8 collation descriptor exits may be specified (in any order). See the *DBA Reference Manual* for more information.

BS2000

Dataset	Link Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1	disk	
Temp area	DDTEMPR1	disk	
Temp overflow (optional)	DDFILEA	disk/tape	Stores descriptor values if the temp dataset is too small
Sort area	DDSORTR1	disk	With large files, split the sort area across two volumes [1]
Sort area	DDSORTR2	disk	
Recovery log (RLOG)	DDRLOGR1	disk	Required when using the recovery log option
Compressed data	DDEBAND	disk/tape	Output of ADACMP or ADAULD utility
ISNs to be deleted	DDISN	disk/tape	ISNs to be deleted [2]
Deleted records	DDOLD	disk/tape	Deleted records, if any [3]
ADARUN parameters	SYSDTA/ DDCARD		*Operations Manual*
ADALOD parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		ADALOD report, see also *Messages and Codes*
ADALOD messages	SYSLST/ DDDRUCK		*Messages and Codes*

Notes:

1. *Performance can be improved when sorting large files if the sort dataset either occupies two volumes, or if two sort datasets are specified. Both datasets must be on the same device type (SORTDEV parameter), and each must be exactly half the size specified by the SORTSIZE parameter.*
2. *Four bytes per ISN, REC–FORM=VB, BUFF–LEN as in sequential file description, REC–SIZE maximum equals BUFF–LEN – 4. (In ISP format, REC–FORM is RECFM; BUFF–LEN is BLKSIZE; and REC–SIZE is LRECL.)*
3. *REC–FORM=VB, BUFF–LEN as in sequential file description, REC–SIZE maximum equals BUFF–LEN – 4. (In ISP format, REC–FORM is RECFM; BUFF–LEN is BLKSIZE; and REC–SIZE is LRECL.)*

ADALOD JCL Example (BS2000)

Load File

In SDF Format:

```
/.ADALOD LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A L O D LOAD FILE
/REMARK *
/ASS-SYSLST L.LOD.LOAD
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
/SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
/SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADALOD,DB=yyyyy,IDTNAME=ADABAS5B
ADALOD LOAD FILE=1
ADALOD NAME= TESTFILE-1
ADALOD MAXISN=10000,DSSIZE=10
ADALOD TEMPSIZE=100,SORTSIZE=50
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADALOD LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A L O D LOAD FILE
/REMARK *
/SYSFILE SYSLST=L.LOD.LOAD
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1,SHARUPD=YES
/FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
/FILE ADAyyyyy.SORT ,LINK=DDSORTR1
/FILE CMP.AUS,LINK=DDEBAND
```

```
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADALOD,DB=yyyyy,IDTNAME=ADABAS5B
ADALOD LOAD FILE=1
ADALOD NAME= TESTFILE-1
ADALOD MAXISN=10000,DSSIZE=10
ADALOD TEMPSIZE=100,SORTSIZE=50
/LOGOFF NOSPOOL
```

Update

In SDF Format:

```
/.ADALOD LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A L O D LOAD FILE
/REMARK *
/DELETE-FILE LOD.ISN
/SET-JOB-STEP
/CREATE-FILE LOD.ISN,PUB(SPACE=(48,48))
/SET-JOB-STEP
/DELETE-FILE LOD.OLD
/SET-JOB-STEP
/CREATE-FILE LOD.OLD,PUB(SPACE=(480,48))
/SET-JOB-STEP
/ASS-SYSLST L.LOD.LOAD
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDDATAR1,ADAyyyyy.DATA,SHARE-UPD=YES
/SET-FILE-LINK DDWORKR1,ADAyyyyy.WORK,SHARE-UPD=YES
/SET-FILE-LINK DDTEMPR1,ADAyyyyy.TEMP
/SET-FILE-LINK DDSORTR1,ADAyyyyy.SORT
/SET-FILE-LINK DDEBAND,CMP.AUS
/SET-FILE-LINK DDISN,LOD.ISN
/SET-FILE-LINK DDOLD,LOD.OLD
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADALOD,DB=yyyyy,IDTNAME=ADABAS5B
ADALOD UPDATE FILE=1,DDISN,SAVEDREC
ADALOD TEMPSIZE=100,SORTSIZE=50
ADALOD DELISN=100 199,230,301 399
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADALOD LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A L O D MASS UPDATE
/REMARK *
/SYSFILE SYSLST=L.LOD.UPDA
/FILE ADA.MOD,LINK=DDLIB
/FILE ADAyyyyy.ASSO ,LINK=DDASSOR1,SHARUPD=YES
/FILE ADAyyyyy.DATA ,LINK=DDDATAR1,SHARUPD=YES
/FILE ADAyyyyy.WORK ,LINK=DDWORKR1,SHARUPD=YES
/FILE ADAyyyyy.TEMP ,LINK=DDTEMPR1
/FILE ADAyyyyy.SORT ,LINK=DDSORTR1
/FILE CMP.AUS,LINK=DDEBAND
/FILE LOD.ISN,LINK=DDISN ,SPACE=(48,48)
/FILE LOD.OLD,LINK=DDOLD ,SPACE=(480,48)
/EXEC (ADARUN,ADA.MOD)
ADARUN PROG=ADALOD,DB=yyyyy,IDTNAME=ADABAS5B
ADALOD UPDATE FILE=1,DDISN,SAVEDREC
ADALOD TEMPSIZE=100,SORTSIZE=50
ADALOD DELISN=100 199,230,301 399
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDDATARn	disk	
Work	DDWORKR1	disk	Required only if Adabas nucleus is not active
Temp area	DDTEMPR1	disk	
Temp overflow (optional)	DDFILEA	disk/tape	Stores descriptor values if the temp dataset is too small
Sort area	DDSORTR1	disk	
Sort area	DDSORTR2	disk	When using large files, split the sort area across two volumes [1]
Recovery log (RLOG)	DDRLOGR1	disk	Required for the recovery log option
Compressed data	DDEBAND	disk/tape	Output of ADACMP or ADAULD utility
ISNs to be deleted	DDISN	disk/tape	ISNs to be deleted [2]
Deleted records	DDOLD	disk/tape	Deleted records, if any [3]
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADALOD parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	ADALOD report, see also *Messages and Codes*
ADALOD messages	DDDRUCK	printer	*Messages and Codes*

Notes:

1. *Performance can be improved when sorting large files if the sort dataset either occupies two volumes, or if two sort datasets are specified. When using two volumes, each volume must be exactly half the size specified by the SORTSIZE parameter. If two datasets are used, both must be on the same device type (SORTDEV parameter).*
2. *Four bytes per ISN, RECFM=VB, BLKSIZE as in sequential file description, LRECL maximum equals BLKSIZE – 4.*
3. *RECFM=VB, BLKSIZE as in sequential file description, LRECL maximum equals BLKSIZE – 4.*

ADALOD JCL Examples (OS/390 or z/OS)

Refer also to ADALODE, ADALODA, ADALODM, and ADALODV in the MVSJOBS dataset for additional ADALOD examples on loading an ADAM file or the Adabas demo files.

Load File

Refer to ADALOD in the MVSJOBS dataset for this example.

```
//ADALOD    JOB
//*
//*    ADALOD : LOAD FILE
//*
//LOD      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDWORKR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.WORKR1 <=== WORK
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1 <=== TEMP
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1 <=== SORT
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.DDEBAND <=== INPUT
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADALOD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADALOD LOAD FILE=1
ADALOD NAME='TESTFILE-1'
ADALOD MAXISN=10000,DSSIZE=10
ADALOD TEMPSIZE=100,SORTSIZE=100
/*
//
```

Update

Refer to ADALODMU in the MVSJOBS dataset for this example.

```
//ADALODMU  JOB
//*
//*      ADALOD : MASS UPDATE
//*
//LOD       EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDASSOR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.ASSOR1 <=== ASSO
//DDDATAR1  DD   DISP=SHR,DSN=EXAMPLE.DByyyyy.DATAR1 <=== DATA
//DDTEMPR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.TEMPR1 <=== TEMP
//DDSORTR1  DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.SORTR1 <=== SORT
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.DDEBAND <=== INPUT
//DDISN     DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.DDISN <=== ISNS TO DEL
//DDOLD     DD   DISP=(NEW,CATLG),DSN=EXAMPLE.DByyyyy.DDOLD, <=== DEL REC
//             SPACE=(TRK,(100,20),RLSE),UNIT=DISK,VOL=SER=VOLvvv
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADALOD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADALOD UPDATE FILE=1,LWP=400K,SAVEDREC
ADALOD TEMPSIZE=100,SORTSIZE=100
ADALOD DELISN=100-199,230,301-399
/*
//
```

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
Associator	DDASSORn	disk	
Data Storage	DDATARn	disk	
Work	DDWORKR1	disk	
Temp area	DDTEMPR1	disk	
Temp overflow (optional)	DDFILEA	disk/tape	Stores descriptor values if temp dataset is too small.
Sort area	DDSORTR1	disk	With large files, split sort area across two volumes [1]
Sort area	DDSORTR2	disk	
Recovery log (RLOG)	DRLOGR1	disk	Required for the recovery log option
Compressed data	DDEBAND	disk/tape	Output of ADACMP or ADAULD utility
ISNs to be deleted	DDISN	disk/tape	ISNs to be deleted [2]
Deleted records	DDOLD	disk/tape	Deleted records, if any [3]
ADARUN parameters	DDCARD	disk/terminal/reader	*Operations Manual*
ADALOD parameters	DDKARTE	disk/terminal/reader	
ADARUN messages	DDPRINT	disk/terminal/printer	ADALOD report, see also *Messages and Codes*
ADALOD messages	DDDRUCK	disk/terminal/printer	*Messages and Codes*

Notes:

1. *Performance can be improved when sorting large files if the sort dataset either occupies two volumes, or if two sort datasets are specified. Both datasets must be on the same device type (SORTDEV parameter), and each must be exactly half the size specified by the SORTSIZE parameter.*
2. *Four bytes per ISN, RECFM=VB, BLKSIZE as in sequential file description, LRECL maximum equals BLKSIZE - 4.*
3. *RECFM=VB, BLKSIZE as in sequential file description, LRECL maximum equals BLKSIZE - 4.*

ADALOD JCL Examples (VM/ESA or z/VM)

Load File

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDEBAND,DSN=FILE001.LODD001,MODE=A
DATADEF DDPRINT,DSN=ADALOD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADALOD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNLOD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=FILE001.LODC001,MODE=A
ADARUN
```

Contents of RUNLOD CONTROL A1:

```
ADARUN PROG=ADALOD,DEVICE=dddd,DB=yyyyy
```

Contents of FILE015 LODC001 A1:

```
ADALOD LOAD FILE=1
ADALOD      NAME='TESTFILE-1'
ADALOD      MAXISN=50000,DSSIZE=10
ADALOD      TEMPSIZE=100,SORTSIZE=50
```

Update

```
DATADEF DDASSOR1,DSN=ADABASVv.ASSO,VOL=ASSOV1
DATADEF DDDATAR1,DSN=ADABASVv.DATA,VOL=DATAV1
DATADEF DDWORKR1,DSN=ADABASVv.WORK,VOL=WORKV1
DATADEF DDTEMPR1,DSN=ADABASVv.TEMP,VOL=TEMPV1
DATADEF DDSORTR1,DSN=ADABASVv.SORT,VOL=SORTV1
DATADEF DDEBAND,DSN=ADALOD.LODD015,MODE=A
DATADEF DDISN,DSN=ADALOD.ISN,MODE=A
DATADEF DDOLD,DSN=ADABASVv.OLDISN,MODE=A
DATADEF DDPRINT,DSN=ADALOD.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADALOD.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNLOD.CONTROL,MODE=A
DATADEF DDKARTE,DSN=UPDATE.CONTROL,MODE=A
ADARUN
```

Contents of RUNLOD CONTROL A1:

```
ADARUN PROG=ADALOD,DEVICE=dddd,DB=yyyyy
```

Contents of UPDATE CONTROL A1:

```
ADALOD UPDATE FILE=1,DDISN,SAVEDREC

ADALOD        TEMPSIZE=100,SORTSIZE=50
ADALOD        DELISN=100-199,230,301-399
```

VSE/ESA

Dataset	Symbolic	Storage	Logical Unit	More Information
Associator	ASSORn	disk		[1]
Data Storage	DATARn	disk		[1]
Work	WORKR1	disk	[1]	Required for inactive nucleus
Compressed data	EBAND	tape disk	SYS010 [1]	
Recovery log (RLOG)	RLOGR1	disk		Required for the recovery log option
Temp area	TEMPR1	disk	[1]	
Temp overflow (optional)	FILEA	tape disk	SYS012 [1]	Stores descriptor values if the temp dataset is too small.
Sort area	SORTR1	disk		With large files, split sort area across two volumes [2]
ISNs to be deleted	ISN	tape disk	SYS016 [1]	ISNs to be deleted
Deleted records	OLD	tape disk	SYS014 [1]	Deleted ISNs
ADALOD messages	—	printer	SYS009	ADALOD report, see also *Messages and Codes*
ADARUN messages	—	printer	SYSLST	*Messages and Codes*
ADARUN parameters	— CARD CARD	reader tape disk	SYSRDR SYS000 [1]	
ADALOD parameters	–	reader	SYSIPT	

Notes:

1. *Any programmer logical unit may be used.*
2. *Performance can be improved when sorting large files if the sort dataset occupies two volumes. When using two volumes, each volume must be exactly half the size specified by the SORTSIZE parameter. If two datasets are used, both must be on the same device type (SORTDEV parameter).*

ADALOD JCS Examples (VSE/ESA)

See appendix B for a description of the VSE/ESA procedures (PROCs).

Load File

Refer to member ADALOD.X for this example.

```
* $$ JOB JNM=ADALOD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADALOD
*       SAMPLE FILE LOAD
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYSTEN,TAPE
// PAUSE MOUNT LOAD INPUT FILE ON TAPE cuu
// TLBL EBAND,'DEMO.FILE'
// MTC REW,SYS010
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADALOD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADALOD LOAD FILE=1
ADALOD NAME='TESTFILE-1'
ADALOD MAXISN=10000,DSSIZE=10
ADALOD TEMPSIZE=100,SORTSIZE=100
/*
/&
* $$ EOJ
```

Update

Refer to member ADALODMU.X for this example.

```
* $$ JOB JNM=ADALODMU,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADALODMU
*       MASS UPDATE
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// ASSGN SYS010,DISK,VOL=DISK01,SHR
// ASSGN SYS014,DISK,VOL=DISK02,SHR
// ASSGN SYS016,DISK,VOL=DISK03,SHR
// DLBL EBAND,'FILE.INPUT',,SD
// EXTENT SYS010,DISK01,1,0,sssss,nnnnn
// DLBL OLD,'FILE.OLD',,SD
// EXTENT SYS014,DISK02,1,0,sssss,nnnnn
// DLBL ISN,'FILE.ISN',,SD
// EXTENT SYS016,DISK03,1,0,sssss,nnnnn
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADALOD,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADALOD UPDATE FILE=1,LWP=400K,SAVEDREC
ADALOD TEMPSIZE=100,SORTSIZE=100
ADALOD DELISN=100-199,230,301-399
/*
/&
* $$ EOJ
```

12

ADAMER : ADAM ESTIMATION

Functional Overview

The ADAMER utility produces statistics that indicate the number of Data Storage accesses required to find and read a record when using an ADAM descriptor. This information is used to determine

- whether usage of the ADAM option would reduce the number of accesses required to retrieve a record using an ADAM descriptor as opposed to the standard Adabas accessing method;
- the amount of Data Storage space required to produce an optimum distribution of records based on the randomization of the ADAM descriptor.

The input data for ADAMER is a dataset containing the compressed records of a file produced by the ADACMP or ADAULD utility.

The field to be used as the ADAM descriptor is specified with the ADAMDE parameter. A multiple value field or a field contained within a periodic group may not be used. The ISN assigned to the record may be used instead of a descriptor as the basis for randomization (ADAMDE=ISN parameter).

The ADAM descriptor must contain a different value in each record, since the file cannot be successfully loaded with the ADAM option of the ADALOD utility if duplicate values are present for the ADAM descriptor. The ADAMER utility requires a descriptor field defined as unique (UQ), but does not check for unique values; checking for unique descriptor values is done by the ADALOD utility when loading the file as an ADAM file.

The BITRANGE parameter may be used to specify that a given number of bits are to be truncated from each ADAM descriptor value before the value is used as input to the randomization algorithm. This permits records containing ADAM descriptor values beginning with the same value (for example, 40643210, 40643220, 40643344) to be loaded into the same physical block in Data Storage. This technique can be used to optimize sequential reading of the file when using the ADAM descriptor to control the read sequence, or to remove insignificant information such as a check digit.

Estimate ADAM Access Requirements

```
ADAMER ADAMDE={ descriptor | ISN }
       MAXISN=maximum-number-of-records
       [BITRANGE={ minimum | 0 } { , maximum | 18 } { , increment | 2 } ]
       [DATADEV={ device-type | ADARUN-device } ]
       [DATAPFAC=padding-factor ]
       [DATASIZE=minimum, maximum [, increment ] ]
       [NOUSERABEND]
       [NUMREC={ number-of-records | all-records } ]
```

Essential Parameters

ADAMDE : ADAM Key

Specifies the descriptor to be used as the ADAM key. If ISN is specified, ADAMER uses the ISN of each input record as input for the randomization algorithm.

The ADAM descriptor must be found in the field definition table (FDT) and be defined as a unique descriptor (UQ). It cannot be a sub-, super-, hyper-, collation, or phonetic descriptor. The descriptor also cannot specify the NU option, cannot be an MU field or a field within a periodic group, and cannot be a variable-length field.

MAXISN : Highest ISN to Be Allocated for the File

The total number of records expected to be contained in the file.

MAXISN should include the number of records to be originally loaded plus the number of records that are likely to be added to the file.

Optional Parameters

BITRANGE : Bit Truncation for ADAM Key

The minimum, maximum, and incremental number of bits to be truncated from each ADAM descriptor value before the value is used as input to the ADAM randomization algorithm. Bits are always truncated from the rightmost portion of the compressed value.

A maximum of 20 different bit truncations is permitted for each ADAMER execution.

Example:

BITRANGE=0,4,2

—results in the truncation of 0 bits, 2 bits, and 4 bits for each Data Storage size for which statistics are provided.

If this parameter is omitted, a default BITRANGE equal to 0,18,2 is used.

DATADEV : Data Storage Device Type

The device type to be used for Data Storage. If DATADEV is not specified, the device type specified by the ADARUN DEVICE parameter is the default.

DATAPFAC : Data Storage Padding Factor

The Data Storage padding factor to be used for the file. The number specified represents the percent of each Data Storage physical block that is not to be used during initial file loading. A value in the range 1–90 may be specified.

If this parameter is omitted, a padding factor of 10 percent is used during ADAMER execution.

DATASIZE : Data Storage Sizes for ADAM Estimates

The Data Storage sizes, in cylinders, for which ADAM statistics are to be provided. A maximum of four Data Storage sizes can be calculated per ADAM execution. The minimum and maximum values may be specified without the increment. ADAMER calculates two increments to produce a report based on all four values.

Example:

DATASIZE=100,175,25

—results in statistics for Data Storage sizes of 100, 125, 150, and 175 cylinders.

If DATASIZE is omitted, ADAMER provides statistics for four Data Storage sizes as follows:

Size 1: The first 100 input records are read and the Data Storage size requirement is based on the ADAM descriptor values present in these records and the value specified for MAXISN. The resulting Data Storage size is used as Data Storage Size 1.

Size 2: Data Storage Size 1 x 1.33.

Size 3: Data Storage Size 2 x 1.33.

Size 4: Data Storage Size 3 x 1.33.

NOUSERABEND : Termination without ABEND

When an error is encountered while the function is running, the utility prints an error message and terminates with user ABEND 34 (with a dump) or user ABEND 35 (without a dump).

If NOUSERABEND is specified, the utility will **not** ABEND after printing the error message. Instead, the message “utility TERMINATED DUE TO ERROR CONDITION” is displayed and the utility terminates with condition code 20.

NUMREC : Maximum Number of Records to Read

The maximum number of records to be read from the input file. If NUMREC is not specified, **all** records are read.

Examples

Example 1:

```
ADAMER ADAMDE=CC, ADAMER DATADEV=3350,DATASIZE=50,110,20,  ADAMER
DATAPFAC=10,MAXISN=225000,BITRANGE=2,6,1
```

The ADAM descriptor is CC. Model 3350 device type is to be used for Data Storage. Statistics for Data Storage sizes of 50, 70, 90, and 110 cylinders are to be provided. Data Storage padding factor of 10 percent is to be used. The planned number of records for the file is 225,000. For each Data Storage size, statistics are to be provided for bit truncations of 2, 3, 4, 5, and 6 bits.

Example 2:

```
ADAMER  ADAMDE=CD,DATADEV=3380,DATAPFAC=5,MAXISN=80000
```

The ADAM descriptor is CD. Model 3380 device type is to be used for Data Storage. Data Storage padding factor of 5 percent is to be used. The planned number of records for the file is 80,000. Default values are to be used for all other parameters.

ADAMER Output Report Description

The following entries appear on the report produced by ADAMER:

Field	Explanation
LOADISNS	Number of records contained in the input dataset.
MAXISN	Total file records (see the MAXISN parameter description).
DATA DEVICE	Data Storage device type (see the DATADEV parameter description).
DATAPFAC	Data Storage padding factor (see DATAPFAC parameter description).

The following fields appear under “AVERAGE NUMBER OF EXCPs”:

Field	Explanation
Data Storage SIZE	See the DATASIZE parameter description. The number of cylinders is rounded up to the nearest integer.
BIT-PARM	See the BITRANGE parameter description.
FOR LOADISNS	The average number of I/Os required to find and read a record when the ADAM descriptor is used. This result assumes that the number of records in the file is equal to the number of records contained in the input dataset.
DISK USAGE	The percentage of Data Storage space occupied after initial loading of the file. This result assumes that the number of records to be loaded is equal to the number of records contained in the input dataset.
FOR MAXISN	The average number of I/Os required to find and read a record when using the ADAM descriptor. This result assumes that the number of records in the file is equal to the value specified with the MAXISN parameter.
DISK USAGE	The percentage of Data Storage space occupied after initial loading of the file. This result assumes that the number of records to be loaded is equal to the number of records specified with the MAXISN parameter.

Using the information contained on the ADAMER report, the user can determine

- the optimum balance between access and Data Storage space requirements; and
- the optimum number of bits that should be truncated from each ADAM descriptor value so that records containing similar beginning values are loaded into the same physical block. This is necessary only if optimization of sequential reading is desired.

JCL/JCS Requirements and Examples

This section describes the job control information required to run ADAMER with BS2000, OS/390 or z/OS, VM/ESA or z/VM, and VSE/ESA systems and shows examples of each of the job streams.

BS2000

Dataset	Link Name	Storage	More Information
Input data	DDEBAND	tape/disk	Output of ADACMP or ADAULD utility
ADARUN parameters	SYSDTA/ DDCARD		*Operations Manual*
ADAMER parameters	SYSDTA/ DDKARTE		
ADARUN messages	SYSOUT/ DDPRINT		*Messages and Codes*
ADAMER messages/report	SYSLST/ DDDRUCK		*Messages and Codes*

ADAMER JCL Example (BS2000)

In SDF Format:

```
/.ADALOD LOGON
/MODIFY-TEST-OPTIONS DUMP=YES
/REMARK *
/REMARK * A D A M E R ALL FUNCTIONS
/REMARK *
/ASS-SYSLST L.MER
/ASS-SYSDTA *SYSCMD
/SET-FILE-LINK DDLIB,ADAvrs.MOD
/SET-FILE-LINK DDASSOR1,ADAyyyyy.ASSO,SHARE-UPD=YES
/SET-FILE-LINK DDEBAND,CMP.AUS
/START-PROGRAM *M(ADA.MOD,ADARUN),PR-MO=ANY
ADARUN PROG=ADAMER,DB=yyyyy,IDTNAME=ADABAS5B
ADAMER ADAMDE=AA,DATASIZE=5200,BITRANGE=8,10,1
ADAMER MAXISN=10000
/LOGOFF SYS-OUTPUT=DEL
```

In ISP Format:

```
/.ADAMER LOGON
/OPTION MSG=FH,DUMP=YES
/REMARK *
/REMARK * A D A M E R ALL FUNCTIONS
/REMARK *
/SYSFILE SYSLST=L.MER
/FILE ADA.MOD,LINK=DDLIB
/FILE CMP.AUS,LINK=DDEBAND
/EXEC (ADARUN,ADA.MOD)


ADARUN PROG=ADAMER,DB=yyyyy,IDTNAME=ADABAS5B
ADAMER ADAMDE=AA,DATASIZE=5200,BITRANGE=8,10,1
ADAMER MAXISN=10000
/LOGOFF NOSPOOL
```

OS/390 or z/OS

Dataset	DD Name	Storage	More Information
Input data	DDEBAND	tape/disk	Output of ADACMP or ADAULD utility
ADARUN parameters	DDCARD	reader	*Operations Manual*
ADAMER parameters	DDKARTE	reader	
ADARUN messages	DDPRINT	printer	*Messages and Codes*
ADAMER messages/report	DDDRUCK	printer	*Messages and Codes*

ADAMER JCL Example (OS/390 or z/OS)

Refer to ADAMER in the MVSJOBS dataset for this example.

```
//ADAMER    JOB
//*
//*   ADAMER:
//*      ADAM ESTIMATION
//*
//MER      EXEC PGM=ADARUN
//STEPLIB   DD   DISP=SHR,DSN=ADABAS.Vvrs.LOAD      <=== ADABAS LOAD
//*
//DDEBAND   DD   DISP=OLD,DSN=EXAMPLE.DByyyyy.COMPR1 <=== COMPRESS DATA
//DDDRUCK   DD   SYSOUT=X
//DDPRINT   DD   SYSOUT=X
//SYSUDUMP  DD   SYSOUT=X
//DDCARD    DD   *
ADARUN PROG=ADAMER,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
//DDKARTE   DD   *
ADAMER MAXISN=1000,ADAMDE=AA,BITRANGE=0,2,4
ADAMER DATADEV=eeee,DATAPFAC=10,DATASIZE=100,175,25
/*
//
```

VM/ESA or z/VM

Dataset	DD Name	Storage	More Information
Input data	DDEBAND	tape/disk	Output of ADACMP or ADAULD utility
ADARUN parameters	DDCARD	disk/terminal/reader	*Operations Manual*
ADAMER parameters	DDKARTE	disk/terminal/reader	
ADARUN messages	DDPRINT	disk/terminal/printer	*Messages and Codes*
ADAMER messages/report	DDDRUCK	disk/terminal/printer	*Messages and Codes*

ADAMER JCL Example (VM/ESA or z/VM)

```
DATADEF DDEBAND,DSN=ADABASVv.BAND,MODE=A
DATADEF DDPRINT,DSN=ADAMER.DDPRINT,MODE=A
DATADEF DUMP,DUMMY
DATADEF DDDRUCK,DSN=ADAMER.DDDRUCK,MODE=A
DATADEF DDCARD,DSN=RUNMER.CONTROL,MODE=A
DATADEF DDKARTE,DSN=ADAMER.CONTROL,MODE=A
ADARUN
```

Contents of RUNMER CONTROL A1:

```
ADARUN PROG=ADAMER,DEVICE=dddd,DB=yyyyy
```

Contents of ADAMER CONTROL A1:

```
ADAMER ADAMDE=AA,DATASIZE=5200,BITRANGE=8,10,1 ADAMER MAXISN=10000
```

VSE/ESA

File	Sym. Name	Storage	Logical Unit	More Information
Input data	EBAND	tape disk	SYS010 *	Output of ADACMP or ADAULD utility
ADARUN parameters	— CARD CARD	reader tape disk	SYSRDR SYS000 *	*Operations Manual*
ADAMER parameters	–	reader	SYSIPT	
ADARUN messages	–	printer	SYSLST	*Messages and Codes*
ADAMER messages/report	–	printer	SYS009	*Messages and Codes*

***** Any programmer logical unit may be used.

ADAMER JCS Example (VSE/ESA)

See appendix B for a description of the VSE/ESA procedures (PROCs). Refer to member ADAMER.X for this example.

```
* $$ JOB JNM=ADAMER,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB ADAMER
// OPTION LOG,PARTDUMP
*       ADAM ESTIMATION
// EXEC PROC=ADAVvLIB
// EXEC PROC=ADAVvFIL
// DLBL EBAND,'EXAMPLE.DByyyyy.COMPR1',0,SD
// EXTENT SYS004
// ASSGN SYS004,DISK,VOL=DISK01,SHR
// EXEC ADARUN,SIZE=ADARUN
ADARUN PROG=ADAMER,MODE=MULTI,SVC=xxx,DEVICE=dddd,DBID=yyyyy
/*
ADAMER MAXISN=1000,ADAMDE=AA,BITRANGE=0,2,4
ADAMER DATADEV=eeee,DATAPFAC=10,DATASIZE=100,175,25
/*
/&
* $$ EOJ
```

A

APPENDIX A : ADABAS SEQUENTIAL FILES

Sequential File Table

This appendix summarizes the sequential files used by the Adabas utilities. Explanations of the table heading and contents are in the text following the table.

Utility	File Name	VSE Tape SYS	Out	In	BLKSIZE by device	Concatenation
ADACDC	DD/SIIN	10		x		Yes
ADACMP	DD/AUSBA	12	x			Yes
	DD/EBAND	10		x		
	DD/FEHL	14	x			
ADALOD	DD/EBAND	10		x		Yes
	DD/FILEA	12	x	x	Yes	
	DD/ISN	16		x		Yes
	DD/OLD	14	x			
ADAMER	DD/EBAND	10		x		
ADAORD	DD/FILEA	10	x	x	Yes	
ADAPLP	DD/PLOG	14		x		Yes
ADAREP	DD/SAVE	10		x		Yes
	DD/PLOG	11		x		Yes
ADARES	DD/BACK	20		x		Yes
	DD/SIAUS1	21	x			
	DD/SIAUS2	22	x			
	DD/SIIN	20		x		Yes
ADASAV	DD/DEL1	31		x		Yes
	DD/DEL2	32		x		Yes
	DD/DEL3	33		x		Yes
	DD/DEL4	34		x		Yes

Utility	File Name	VSE Tape SYS	Out	In	BLKSIZE by device	Concatenation
	DD/DEL5	35		x		Yes
	DD/DEL6	36		x		Yes
	DD/DEL7	37		x		Yes
	DD/DEL8	38		x		Yes
	DD/DUAL1	21	x			
	DD/DUAL2	22	x			
	DD/DUAL3	23	x			
	DD/DUAL4	24	x			
	DD/DUAL5	25	x			
	DD/DUAL6	26	x			
	DD/DUAL7	27	x			
	DD/DUAL8	28	x			
	DD/FULL	30		x		Yes
	DD/PLOG	10		x		Yes
	DD/REST1	11		x		Yes
	DD/REST2	12		x		
	DD/REST3	13		x		
	DD/REST4	14		x		
	DD/REST5	15		x		
	DD/REST6	16		x		
	DD/REST7	17		x		
	DD/REST8	18		x		
	DD/SAVE1	11	x			
	DD/SAVE2	12	x			
	DD/SAVE3	13	x			
	DD/SAVE4	14	x			

Utility	File Name	VSE Tape SYS	Out	In	BLKSIZE by device	Concatenation
	DD/SAVE5	15	x			
	DD/SAVE6	16	x			
	DD/SAVE7	17	x			
	DD/SAVE8	18	x			
ADASEL	DD/EXPA1	11	x			
	DD/EXPA2	12	x			
	DD/EXPA3	13	x			
	DD/EXPA4	14	x			
	DD/EXPA5	15	x			
	DD/EXPA6	16	x			
	DD/EXPA7	17	x			
	DD/EXPA8	18	x			
	DD/EXPA9	19	x			
	DD/EXPA10	20	x			
	DD/EXPA11	21	x			
	DD/EXPA12	22	x			
	DD/EXPA13	23	x			
	DD/EXPA14	24	x			
	DD/EXPA15	25	x			
	DD/EXPA16	26	x			
	DD/EXPA17	27	x			
	DD/EXPA18	28	x			
	DD/EXPA19	29	x			
	DD/EXPA20	30	x			
	DD/SIIN	10		x		Yes

Utility	File Name	VSE Tape SYS	Out	In	BLKSIZE by device	Concatenation
ADAULD	DD/OUT1	10	x		Yes	
	DD/OUT2	11	x		Yes	
	DD/ISN	12	x		Yes	
	DD/SAVE	13		x		Yes
	DD/PLOG	14		x		Yes
	DD/FULL	30		x		Yes
	DD/DEL1–8	31–38		x		Yes
ADAVAL	DD/FEHL	10	x		Yes	

Files that are both output and input are first written and then read by the indicated program. BS2000, VM/ESA or z/VM, OS/390 or z/OS, and OS-compatible files have “DD...” names (DDSIIN, DDFEHL, etc.); VSE/ESA file names are without “DD”.

Operating System Dependencies

The following sections describe characteristics of file and device definition by operating system.

BS2000 Systems

Note:
This discussion uses SPF format. In ISP format:

SPF Format	*ISP Format*
BUFF–LEN	*BLKSIZE defined by BLKSIZE=(STD,16)*
REC–FORM	*RECFM*
REC–SIZE	*RECSIZE*
SET–FILE–LINK	*FILE*

The LINK name by which a file is referenced is determined as follows:

- The characters DD are prefixed to the file name to form the LINK name.
- If files for which the column “Concatenation” contains “Yes” are on tape, they may be concatenated as follows: the first file is read using the indicated LINK name; at the first end-of-file, 01 is appended to the LINK name; and, if there is a /SET–FILE–LINK (in ISP format /FILE) statement for that LINK name, reading continues.
- Each subsequent end-of-file adds 1 to the LINK name, and as long as there is a /SET–FILE–LINK (in ISP format /FILE) statement for that LINK name, reading continues through a maximum of 99. For LINK names longer than six characters, the excess characters will be overlaid with the file number increment (e.g., DDEBAND becomes DDEBAN01).
- BS2000 does not support the backward reading of multivolume tape files; therefore, all volumes of the ADARES DDBACK file must be specified in the reverse order in which they were written on /SET–FILE–LINK (in ISP format /FILE) statements using the LINK names DDBACK, DDBACK01, DDBACK02, and so on.

The BUFF–LEN of a sequential file is determined as follows:

1. The BUFF–LEN is obtained from the /SET–FILE–LINK statement or the dataset's catalog entry, if present.
2. If the BUFF–LEN cannot be obtained from the /SET–FILE–LINK statement and/or catalog, the value of the ADARUN QBLKSIZE parameter is used, if specified.
3. Otherwise, the BUFF–LEN depends on the device type as follows:

Tape:	32760
Disk:	32768 (BUFF–LEN=(STD,16))

The REC–SIZE and REC–FORM should be as follows:

Tape:	REC–SIZE = BUFF–LEN – 4; REC–FORM = V;
Disk:	REC–SIZE = BUFF–LEN – 20; RECFORM = V;
Input:	Obtained from the /SET–FILE–LINK statement or the dataset's catalog entry.

Note:
Do not specify REC–FORM, REC–SIZE, or BUFF–LEN for input datasets unless the TAPE dataset contains no REC–FORM, REC–SIZE, or BUFF–LEN values in HDR2.

The SPACE parameter for primary and secondary allocations must specify a multiple of three (3) times the number of PAM blocks specified in the BUFF–LEN parameter. Otherwise, I/O errors will occur. For the default /CREATE–FILE ...,PUB(SPACE(48,48)) and /SET–FILE–LINK ...,BUFF–LEN=STD(16) (in ISP format, BLKSIZE=(STD,16), SPACE=(48,48)) is the smallest valid value.

The portions of the DDDRUCK and DDPRINT datasets already written to disk can be accessed during either a regular nucleus or utility session for reading. This includes the following BS2000 read accesses:

- SHOW–FILE
- @READ dataset
- /COPY–FILE (in ISP format, /COPY)

Concatenation of Sequential Input Files for BS2000

For using more than one dataset as input medium to an ADABAS utility, some operating systems (such as OS/390) provide a concatenation feature.

For BS2000 this feature is simulated by adding /SET–FILE–LINK (in ISP format, /FILE) statements with modified LINK names created from the original and a two-digit increment (ranging from 01 to 99):

```
/SET-FILE-LINK DDTEST,firstfile
/SET-FILE-LINK DDTEST01,secondfile
/SET-FILE-LINK DDTEST02,thirdfile
...
/SET-FILE-LINK DDTEST99,lastfile
```

In ISP format:

```
/FILE firstfile ,LINK=DDTEST
/FILE secondfile,LINK=DDTEST01
/FILE thirdfile ,LINK=DDTEST02
...
/FILE lastfile ,LINK=DDTEST99
```

For those original LINK names that are 7 or 8 characters long, the incremental number occupies the 7th and 8th position. For example:

```
/SET-FILE-LINK DDEBAND,firstfile
/SET-FILE-LINK DDEBAND01,lastfile
```

In ISP format:

```
/FILE firstfile ,LINK=DDEBAND
/FILE secondfile,LINK=DDEBAN01
```

When processing input files that have the concatenation option at end-of-file of one input file, a check is made to determine whether a /SET–FILE–LINK (in ISP format, /FILE) statement exists for the next dataset. If none exists, the sequential GET call returns EOF; otherwise, the dataset currently open is closed, and an open is tried for the next file.

Files concatenated in this way must have the same file characteristics (block size, record format and record size).

This concatenation feature applies also to files that are processed backwards. The order of the LINK names is the reverse of the creation order. For example, ADARES with DDBACK:

```
/SET-FILE-LINK DDBACK,lastfile
/SET-FILE-LINK DDBACK01,filebeforelast
/...
/SET-FILE-LINK DDBACKnn,firstfile
```

In ISP format:

```
/FILE lastfile ,LINK=DDBACK
/FILE filebeforelast,LINK=DDBACK01
/....
/FILE firstfile ,LINK=DDBACKnn
```

Note that this feature can also be used to process a multivolume file backwards, if each volume is specified with a separate /SET–FILE–LINK (in ISP format, /FILE) statement.

The following list is of LINK names/utilities with the concatenation option:

DDDELn (where n = 1–8)	ADASAV
DDEBAND	ADACMP ADALOD ADAMER
DDFULL	ADASAV
DDISN	ADALOD
DDPLOG	ADAPLP ADASAV
DDBACK	ADARES
DDSIIN	ADARES ADASEL
DDREST1	ADASAV (LINK names used are DDREST1, DDREST01, DDREST02, and so on.)

Example for Use of the Concatenation Feature with ADARES

During the last nucleus session, three protection log files were produced with ADARES PLCOPY named F1, F2, F3.

When backing out the session to a specific point, use the following /SET–FILE–LINK (in ISP format, /FILE) statements for the ADARES BACKOUT function:

```
/SET-FILE-LINK DDBACK,F3
/SET-FILE-LINK DDBACK01,F2
/SET-FILE-LINK DDBACK02,F1
```

In ISP format:

```
/FILE F3,LINK=DDBACK
/FILE F2,LINK=DDBACK01
/FILE F1,LINK=DDBACK02
```

To regenerate the database from the protection log that was produced during the session, use the following /SET–FILE–LINK (in ISP format, /FILE) statements for the ADARES REGENERATE function:

```
/SET-FILE-LINK DDSIIN,F1
/SET-FILE-LINK DDSIIN01,F2
/SET-FILE-LINK DDSIIN02,F3
```

In ISP format:

```
/FILE F1,LINK=DDSIIN
/FILE F2,LINK=DDSIIN01
/FILE F3,LINK=DDSIIN02
```

Control Statement Read Procedure in Version 11.2 (OSD 2.0)

With BS2000 version 11.2 (OSD 2.0), the SYSIPT system file is no longer available. Beginning with version 5.3.3, ADABAS can read all control statements from the SYSDTA system file.

When running on BS2000 Versions 10.0 or 11.0, the SYSIPT assignment can still be used; however, Software AG recommends adapting all ADABAS utility and Entire Net-Work job control to indicate the SYSDTA system file before migrating to BS2000 version 11.2 (OSD 2.0).

ADARUN TAPEREL : Tape Release Option

The ADARUN parameter TAPEREL is required to perform the tape handling control for utilities that access files on tape. See the *ADABAS Operations Manual* for more information.

OS/390 or MVS/ESA Systems

The DDNAME is formed by prefixing the characters DD to the file name.

To allow utilities to access dataset information after closing, the DD statement for sequential datasets used in utilities should not contain FREE=CLOSE.

The BLKSIZE of a sequential file is determined as follows:

- If the column, “BLKSIZE by device” specifies Yes for a file, the default BLKSIZE depends on the device type as follows:

Tape:	32760
3330 disk:	13030
3340 disk:	8368
3350 disk:	19069
3375 disk:	17600
3380 disk:	23476
3390 disk:	27998

- If the column “BLKSIZE by device” does not specify Yes for a file, the file’s BLKSIZE is obtained from the DD statement or dataset label, if present. It must be present for any input file.
- If the column “BLKSIZE by device” does not specify Yes for a file and the BLKSIZE cannot be obtained from the DD statement or dataset label, the value of the ADARUN QBLKSIZE parameter is used, if specified.

Except for ADACMP EBAND, the RECFM and LRECL of all sequential files are VB and BLKSIZE–4, respectively. For ADACMP EBAND, RECFM and LRECL must be available from the DD statement and/or dataset label.

If the DCB BUFNO parameter is not provided on the DD statement, the operating system default will be used.

VM/ESA Systems

The DATADEF name is formed by prefixing the characters DD to the file name.

The BLKSIZE of a sequential file is determined as follows:

- If the column, “BLKSIZE by device” specifies Yes for a file, the BLKSIZE depends on the device type as follows:

Tape:	32760
FBA disk:	32760
3330 disk:	13030
3340 disk:	8368
3350 disk:	19069
3375 disk:	17600
3380 disk:	23476
3390 disk:	27998

- If the column “BLKSIZE by device” does not specify Yes for a file, the file’s BLKSIZE is obtained from the DD statement or dataset label, if present. It must be present for any input file.
- If the column “BLKSIZE by device” does not specify Yes for a file and the BLKSIZE cannot be obtained from the DD statement or dataset label, the value of the ADARUN QBLKSIZE parameter is used, if specified.

For all sequential files except ADACMP EBAND, the RECFM is VB and LRECL is (BLKSIZE – 4). For ADACMP EBAND, RECFM and LRECL must be available from the DATADEF statement and/or dataset label.

VSE/ESA Systems

The following items determine how a file is referenced by the utilities running under VSE/ESA:

- The file name is used as the filename on the DLBL or TLBL statement.
- If files for which the column “Concatenation” contains Yes are on tape, they may be concatenated as follows:
  - The file is first read using the indicated file name.
  - At the first end-of-file, 01 is appended to the file name and, if there is a TLBL statement for that filename, reading continues.
  - At each subsequent end-of-file, 1 is added to the file name and reading continues as long as there is a TLBL statement for that filename, up through a maximum of 99.
- Since VSE does not support reading multivolume tape files backward, each volume of the ADARES BACK file must be specified in reverse order from the way it was written on TLBL statements using the filenames BACK, BACK01, BACK02, and so on.

Any programmer logical unit may be used for sequential files on disk. The **VSE Tape SYS** number must be used for sequential files on tape; any or all of these numbers may be changed using procedures defined in the *ADABAS Installation Manual*.

The BLKSIZE of a sequential file is determined as follows:

- If the column “BLKSIZE by Device” specifies Yes for a file, the BLKSIZE depends on the device type as follows:

Tape:	32760
FBA disk:	32760
3330 disk:	13030
3340 disk:	8368
3350 disk:	19069
3375 disk:	17600
3380 disk:	23476
3390 disk:	27998

- If the column “BLKSIZE by Device” does not specify Yes for a file, the value of the ADARUN QBLKSIZE parameter is used, if specified.

For ADACMP EBAND, this BLKSIZE is checked and may then be changed to an actual BLKSIZE, depending on the RECFM and LRECL parameters as specified on ADACMP control cards, as follows:

If RECFM= ...	then the actual BLKSIZE= ...
F	LRECL.
FB	BLKSIZE/LRECL*LRECL, where the remainder of the division is discarded before the multiplication.
U	LRECL, which must not be greater than BLKSIZE.
V	LRECL+4, which must not be greater than BLKSIZE.
VB	BLKSIZE, which must not be less than LRECL+4.

The RECFORM of all sequential files except ADACMP EBAND is VARBLK. For ADACMP EBAND, it is provided by the RECFM parameter of a control statement.

To distinguish whether VSE message 4140D refers to the first or a subsequent volume of a multivolume tape file, message ADAI31 is written to the operator whenever a tape file is opened, but not at end-of-volume.

Concatenation of Sequential Input Files for VSE/ESA

In those cases where it is desired to use more than one dataset as input medium for an ADABAS utility, a concatenation feature is provided by some operating systems (OS/390 or z/OS, for example).

For VSE, this feature is simulated by adding FILE statements with modified LINK names created from the original and a two-digit increment (ranging from 01 to 99):

```
// DLBL TEST  ,'firstfile'
// EXTENT ...
// DLBL TEST01,'secondfile'
// EXTENT ...
...
// DLBL TEST99,'lastfile'
// EXTENT ...
```

When processing input files that have the concatenation option at end-of-file (EOF) of one input file, a check is made to determine whether a FILE statement exists for the next dataset. If it does not exist the Sequential Get call returns EOF; otherwise, the dataset currently open is closed and an open is tried for the next file.

Files concatenated in this way must have the same file characteristics (block size, record format, and record size).

This concatenation feature applies also to files that are processed backwards. The order of the LINK names is the reverse of the creation order; for example, ADARES with BACK:

```
// DLBL BACK  ,'lastfile'
// EXTENT ...
// DLBL BACK01,'filebeforelast'
// EXTENT ...
...
// DLBL BACKnn,'firstfile'
// EXTENT ...
```

Note that this feature could also be used to process a multivolume file backwards, if each volume is specified with a separate FILE statement.

The following are the LINK names/utilities with the concatenation option:

DELn (where n=1–8)	ADASAV
EBAND	ADACMP ADALOD ADAMER
FULL	ADASAV
ISN	ADALOD
PLOG	ADAPLP ADASAV
BACK	ADARES
SIIN	ADARES ADASEL
REST1	ADASAV (LINK names used are REST1, REST101, REST102, and so on.)

Example for Use of the Concatenation Feature with ADARES

During the last nucleus session, three protection log files were produced with ADARES PLCOPY named F1, F2, F3.

When deciding to back out the session to a specific point, the following FILE statements should be used for the ADARES BACKOUT function:

```
// DLBL BACK  ,'F3'
// EXTENT ...
// DLBL BACK01,'F2'
// EXTENT ...
// DLBL BACK02,'F1'
// EXTENT ...
```

To regenerate the database from the protection log that was produced during the session, the following FILE statements should be used for the ADARES REGENERATE function:

```
// DLBL SIIN  ,'F1'
// EXTENT ...
// DLBL SIIN01,'F2'
// EXTENT ...
// DLBL SIIN02,'F3'
// EXTENT ...
```

B

APPENDIX B : PROCEDURES FOR VSE/ESA EXAMPLES

The VSE/ESA examples assume that the procedures for defining Adabas libraries (ADAVvLIB) and Adabas files (ADAVvFIL) have been cataloged into an accessible procedure library.

For information about cataloging these procedures, refer to the section **Catalog Procedures for Defining Libraries and the Database** in the VSE/ESA chapter of the *Adabas Installation Manual*.

Information about cataloging procedures for use with the Delta Save Facility are documented in the *Adabas Delta Save Facility Manual*.

Adabas Libraries (ADAVvLIB)

```
// PROC
* ******************************************************* *
* LIBRARY DEFINITIONS AND CHAINING FOR ADABAS              *
* ******************************************************* *
// SETPARM VERS=vrs                  <- CURRENT VERSION
// SETPARM ADALIB=SAGLIB             <- SAG PRODUCT LIBRARY
// SETPARM ADASUB=ADA&VERS           <- ADABAS SUBLIBRARY
// DLBL SAGLIB,'SAG.PRODUCT.LIBRARY'
// EXTENT ,vvvvvv
// LIBDEF *,SEARCH=&ADALIB..&ADASUB,TEMP
// LIBDEF PHASE,CATALOG=&ADALIB..&ADASUB,TEMP
// ASSGN SYS009,PRINTER
```

—where

vrs	is the Adabas version, revision, and system maintenance (SM) level
vvvvvv	is the programmer logical unit assigned

Adabas Files (ADAVvFIL)

```
// ASSGN SYS031,dddd,VOL=ADA001,SHR
// ASSGN SYS032,dddd,VOL=ADA002,SHR
// ASSGN SYS033,dddd,VOL=ADA003,SHR
// ASSGN SYS034,dddd,VOL=ADA004,SHR
// DLBL ASSOR1,'EXAMPLE.ADAyyyyy.ASSOR1',99/365,DA
// EXTENT SYS031,ADA001,,,15,1500
// DLBL DATAR1,'EXAMPLE.ADAyyyyy.DATAR1',99/365,DA
// EXTENT SYS032,ADA002,,,15,3000
// DLBL WORKR1,'EXAMPLE.ADAyyyyy.WORKR1',99/365,DA
// EXTENT SYS033,ADA003,,,15,600
// DLBL PLOGR1,'EXAMPLE.ADAyyyyy.PLOGR1',99/365,DA
// EXTENT SYS034,ADA004,,,15,600
// DLBL PLOGR2,'EXAMPLE.ADAyyyyy.PLOGR2',99/365,DA
// EXTENT SYS034,ADA004,,,615,600
// DLBL CLOGR1,'EXAMPLE.ADAyyyyy.CLOGR1',99/365,DA
// EXTENT SYS034,ADA004,,,1215,750
// DLBL CLOGR2,'EXAMPLE.ADAyyyyy.CLOGR2',99/365,DA
// EXTENT SYS034,ADA004,,,1965,750
// DLBL TEMPR1,'EXAMPLE.ADAyyyyy.TEMPR1',99/365,DA
// EXTENT SYS032,ADA002,,,3015,1500
// DLBL SORTR1,'EXAMPLE.ADAyyyyy.SORTR1',99/365,DA
// EXTENT SYS033,ADA003,,,615,375
// EXTENT SYS034,ADA004,,,2715,375
// DLBL RLOGR1,'EXAMPLE.ADAyyyyy.RLOGR1',99/365,DA
// EXTENT SYS033,ADA003,,,990,150
```

C

APPENDIX C : SUPPLIED UES ENCODINGS

The tables in this appendix list the encodings available with universal encoding support (UES) when executing Adabas utilities or issuing Adabas commands. Encodings represent single-byte character sets (Latin–1 or not) or double-/multiple-character sets. If encodings

- have a character set in common with other encodings so that conversion between them is accomplished without loss of data, they are “interoperable”.
- are not interoperable, they are “coexistent” with other encodings.

Columns used in the following tables are described as follows:

Key	Entire Conversion Services (ECS) key number in decimal and hexadecimal	
CS	character set as identified by IBM	
CP	code page usually identical to the “key”. It is **not** identical when • CP does not fit into the ECS key number range 1–4095; or • the encoding is constructed from two or more code pages.	
Size	F	encoding uses all allocated graphical character space
	M	maximum for given ESID
	S	subset
ESID	encoding scheme identifier with the following entries:	
	1100	EBCDIC fixed 1
	1301	EBCDIC mixed DBCS modal
	2100	IBM PC data fixed 1
	2200	IBM PC data DBCS only
	2300	IBM PC data mixed DBCS nonmodal
	4100	ISO–8 fixed 1
	4105	ISO–8 fixed 1 graphic characters in the range x’80’– x‘9F’ reserved for control codes
Fill	the hexadecimal value representing the fill character used in the encoding.	
Sub	the hexadecimal value representing the substitution character used in the encoding.	

Interoperable Encodings

Single-Byte Character Sets (Latin–1)

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
37	025	697	37	F 190	1100	40	3F	CECP: USA, Canada (ESA*), Netherlands, Portugal, Brazil, Australia, New Zealand
273	111	697	273	F 190	1100	40	3F	CECP: Austria, Germany, de_deu
277	115	697	277	F 190	1100	40	3F	CECP: Denmark, Norway
278	116	697	278	F 190	1100	40	3F	CECP: Finland, Sweden
280	118	697	280	F 190	1100	40	3F	CECP: Italy, it_ita
284	11C	697	284	F 190	1100	40	3F	CECP: Spain, Latin America (Spanish), es_esp
285	11D	697	285	F 190	1100	40	3F	CECP: United Kingdom, en_gbr
297	129	697	297	F 190	1100	40	3F	CECP: France, fr_fra
500	1F4	697	500	F 190	1100	40	3F	CECP: Belgium, Canada (AS/400*), Switzerland, international Latin–1
819	333	697	819	F 190	4100	20	1A	ISO 8859–1: Latin alphabet, Latin–1
871	367	697	871	F 190	1100	40	3F	CECP: Iceland, is_ISL
923	39B	**	923	F 190	4100	20	1A	ISO 8859–15 with euro sign
924	39C	**	924	F 190	1100	40	3F	IBM EBCDIC with same char. set as 923.
1140	474	**	1140	F 190	1100	40	3F	IBM EBCDIC 37 with euro sign
1141	475	**	1141	F 190	1100	40	3F	IBM EBCDIC 273 with euro sign
1142	476	**	1142	F 190	1100	40	3F	IBM EBCDIC 277 with euro sign
1143	477	**	1143	F 190	1100	40	3F	IBM EBCDIC 278 with euro sign
1144	478	**	1144	F 190	1100	40	3F	IBM EBCDIC 280 with euro sign
1145	479	**	1145	F 190	1100	40	3F	IBM EBCDIC 284 with euro sign
1146	47A	**	1146	F 190	1100	40	3F	IBM EBCDIC 285 with euro sign
1147	47B	**	1147	F 190	1100	40	3F	IBM EBCDIC 297 with euro sign
1148	47C	**	1148	F 190	1100	40	3F	IBM EBCDIC 285 with euro sign

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
1149	47D	**	1149	F 190	1100	40	3F	IBM EBCDIC 871 with euro sign
*3946	F6A	697	850	S 190	2100	20	7F	PC data–190: Latin alphabet, Latin–1 (CCSID=4946)

***** This code page is not yet available but may be at a later time.

******This information is not yet available.

Single-Byte Character Sets (Non-Latin–1)

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
420	1A4	235	420	M 181	1100	40	3F	IBM, Arabic (all presentation shapes)
424	1A8	941	424	M 152	1100	40	3F	IBM, Hebrew, iw_isr
813	32D	925	813	M 183	4100	20	1A	ISO 8859–7: Greek/Latin, el_GRC
870	366	959	870	F 190	1100	40	3F	IBM, Latin–2
875	36B	925	875	M 184	1100	40	3F	IBM, Greek, el_GRC
912	390	959	912	F 190	4100	20	1A	ISO 8859–2: Latin–2
914	392		914	F 190	4100	20	1A	ISO 8859–4
915	393	1150	915	F 190	4100	20	1A	ISO 8859–5: Cyrillic, 8-bit
916	394	941	916	M 152	4100	20	1A	ISO 8859–8: Hebrew, iw_ISR
918	396	1160	918	F 190	1100	40	3F	IBM, Urdu
919	397		919	F190	4100	20	1A	ISO 8859–10
920	398	1152	920	F 190	4100	20	1A	ISO 8859–9 Latin–5 (ECMA–128, Turkey TS–5881), tr_TUR
921	399	1305	921	F 190	4100	20	1A	IBM, Baltic, 8-bit
922	39A	1307	922	F 190	4100	20	1A	IBM, Estonia, 8-bit
1006	3EE	1160	1006	F 190	4100	20	1A	IBM, Urdu
1025	401	1150	1025	F 190	1100	40	3F	IBM, Cyrillic multilingual
1026	402	1152	1026	F 190	1100	40	3F	IBM, Turkey Latin–5, tr_TUR
1112	458	1307	1112	F 190	1100	40	3F	IBM, Baltic, multilingual (EBCDIC)
1122	462	1307	1122	F 190	1100	40	3F	IBM, Estonia (EBCDIC)

Double- and Multiple-Byte Character Sets

Key		CS	CP	F/M/S Size	ESID	Fill	Sub	Description
Dec	Hex							
290	122	1172	290	M 162	1100	40	3F	Japanese Katakana host extended SBCS, ja_JPN
836	344	1174	836	M 99	1100	40	3F	Simplified Chinese host extended SBCS (EBCDIC), zh_CHN
1027	403	1172	1027	M 162	1100	40	3F	IBM, Japanese Latin host extended SBCS, ja_JPN
1041	411	1172	1041	S 162	2100	20	1A	IBM, Japanese PC data extended SBCS, ja_JPN
1043	413	1175	1043	M 97	2100	20	1A	IBM, traditional Chinese PC data extended SBCS, zh_TWN
1115	45B	1174	1115	M 99	2100	20	1A	IBM, simplified Chinese PC data single-byte (IBM GB) including 5 SAA SB characters, zh_CHN
1380	564	937	1380	M 9355	2200	A1A1	FEFE	IBM, simplified Chinese DBCS PC (IBM GB) including 1880 UDC and 31 IBM-selected, zh_CHN
1381	565	1174 937	1115 1380	M 9454	2300	20 A1A1	1A FEFE	IBM, simplified Chinese PC data mixed (IBM GB) including 1880 UDC, 31 IBM-selected and 5 SAA SB characters, zh_CHN
3026	BD2	1172 370	290 300	S 9306	1301	40 4040	3F FEFE	IBM CCSID 5026, Japanese Katakana-Kanji host mixed including 1880 UDC, extended SBCS, ja_JPN
3035	BDB	1172 370	1027 300	S 9306	1301	40 4040	3F FEFE	IBM CCSID 5035, Japanese Latin-Kanji host mixed including 1880 UDC, extended SBCS, ja_JPN
3396	D44	370	300	S 9144	1200	4040	FEFE	IBM, CCSID 4396, Japanese host double-byte including 1880 UDC, ja_JPN

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
3709	E7D	1175	3709	S 97	1100	40	3F	IBM CCSID 28709: traditional Chinese host extended SBCS (EBCDIC), zh_TWN
4091	FFB			F 63456		0020	001A	UTF–8: with user/compatibility area; without surrogates

Coexistent Encodings

Single-Byte Character Sets

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
3	3	103	367	F 94	5100	20	1A	US–ASCII
6	6			F 190	4100	20	1A	ISO 8859–3, Latin–3, Afrikaans, Catalan, Dutch, English, German, Italian, Maltese, Spanish, Turkish
9	9			M 145	4100	20	1A	ISO 8859–6, Arabic
437	1B5	697	437	F 222	2100	20	1A	Microsoft MSDOS US, en_USA
850	350	1106	950	F 222	2100	20	1A	Microsoft multilingual code page 850, Latin–1
858	350	1106	950	F 222	2100	20	1A	Microsoft multilingual code page 850, Latin–1, euro-ready
1250	4E2	1400	1250	M 217	4105	20	1A	Microsoft, Windows ANSI, Latin–2, euro-ready
1251	4E3	1401	1251	M 220	4105	20	1A	Microsoft, Windows ANSI, Cyrillic, euro-ready
1252	4E4	1402	1252	M 215	4105	20	1A	Microsoft, Windows ANSI, Latin–1, euro-ready
1253	4E5	1403	1253	M 206	4105	20	1A	Microsoft, Windows ANSI, Greek, euro-ready
1254	4E6	1404	1254	M 215	4105	20	1A	Microsoft, Windows ANSI, Turkey, euro-ready

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
1255	4E7	1405	1255	M 194	4105	20	1A	Microsoft, Windows ANSI, Hebrew, euro-ready
1256	4E8	1406	1256	M 214	4105	20	1A	Microsoft, Windows ANSI, Arabic, euro-ready
1257	4E9	1407	1257	M 203	4105	20	1A	Microsoft, Windows ANSI, Baltic rim, euro-ready
2084	824			M 222	3100	20	1A	RELCOM, koi8–r, RFC–1489, Russian internet character set
2087	827		775	M 222	3100	20	1A	Baltic
2258	8D2		1258	M 222	4105	20	1A	Microsoft, Windows ANSI, Vietnam
4092	FFC			3	–	20	A1	Software AG, used for ECS internally
4093	FFD				4100	20	1A	Software AG, old TS default, ASCII
4094	FFE	–	–		1100	40	3F	Software AG, old TS default, EBCDIC
4095	FFF			–	7200	0020	001A	Unicode

Double- and Multiple-Byte Character Sets

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
18	12	103 1061 1121 1062	367 952 896 953	S 13102	4403	20 8140	1A	IBM CCSID 1350, EUC–JP, ja_JPN composed of US–ASCII, JIS–X–0208, HW Katakana, and JIS–X–0212–90
36	24				5404	20	1A	ISO–2022–JP, US–ASCII, JIS–Roman, JIS–0208–1983, JIS_C_6226–1978
300	12C	1001	300	M 11634	1200	4040	FEFE	IBM Japanese Latin host double-byte including 4370 UDC, ja_JPN
301	12D	370	301	M 9144	2200	8140	FCFC	IBM Japanese PC double-byte including 1880 UDC, SJIS, ja_JPN

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
835	343	935	835	M 20263	1200	4040	FEFE	IBM traditional Chinese host double-byte including 6204 UDC (EBCDIC) zh_TWN
837	345	937	837	M 9355	1200	4040	FEFE	Simplified Chinese host double-byte including 1880 UDC (EBCDIC), zh_CHN
927	39F	935	927	M 20263	2200	8140	FCFC	IBM traditional Chinese PC data double-byte including 6204 UDC, zh_TWN
932	3A4	11223 70	897 301	M 9301	2300	20 8140	1A	Microsoft, JIS Roman, JIS–X–208, half-width Katakana, ja_JPN
935	3A7	1174 937	836 837	M 9454	1301	40 4040	3F FEFE	Simplified Chinese host mixed including 1880 UDC, extended SBCS (EBCDIC), zh_CHN
936	3A8	1185 937	1185 937	M 9449	2300	20 1A1A	1A	Microsoft, GB Roman, GB 2312–80, zh_CHN
937	3A9	1175 935	903 937	S 20360	1301	40 4040	3F FEFE	IBM traditional Chinese host mixed including 6204 UDC, extended SBCS (EBCDIC), zh_TWN
942	3AE	1172 370	1041 301	M 9306	2300	20 8140	1A FCFC	IBM, Japanese PC data mixed including 1880 UDC, extended SBCS, ja_JPN
948	3B4	1175 935	948	M 20360	2300	20 8140	1A FCFC	IBM, traditional Chinese PC data mixed including 6204 UDC, extended SBCS, zh_TWN
949	3B5	1278 1050	949	M 10197	2300	20 A1A1	1A AFFE	IBM, Korean IBM KS code – PC data mixed including 1880 UDC, ko_KOR
950	3B6	103 935	950	M 20357	2300	20 A140	1A	Microsoft, Big Five, zh_TWN

Key		CS	CP	F/M/S	ESID	Fill	Sub	Description
Dec	Hex			Size				
951	3B7	1050	951	M 10103	2200	A1A1	AFFE	IBM, Korean IBM KS code – PC data double-byte including 1880 UDC, ko_KOR
3001		10001	3001			20 8140	1A	MAC Japanese, JIS–Roman, JIS–X–208, HW Katakana, ja_JPN

INDEX

A

ADAACK utility, 11–18
  ACCHECK function, 12
  BS2000 JCL, 14
  functional overview, 11
  VSE/ESA JCS, 18
  z/OS or OS/390 JCL, 16
  z/VM or VM/ESA JCL, 17
Adabas control block
  start logging, using utility, 192
  stop logging, using utility, 192
Adabas Delta Save Facility, display status, using utility, 187
Adabas file, create, using utility, 60
Adabas files, sequential, list by utility, 385
Adabas Review
  deactivate, using utility, 195
  hub ID, set/modify, using utility, 195
  local mode, switch to using utility, 195
Adabas Statistics Facility, using ADADBS REFRESHSTATS with, 201
ADACDC utility, 19–41
  examples, 33
  extract file, 20
  functional overview, 19
  input data, 22
  JCL requirements and examples, 34
    BS2000, 34
    VSE/ESA, 40
    z/OS or OS/390, 36
    z/VM or VM/ESA, 38
  operating system factors, 27
    BS2000, 28
    VSE/ESA, 28
    z/OS or OS/390, 27
  operation, 19
  output data, 22
  parameters, 25
  phases of operation, 19
  primary output file, 20
    checkpoints, 21
  running, 25
  syntax, 25
  transaction file, 23
  user exit, 29
    calls, 31
    installing, 29
    interface description, 29
    using to update or add records, 33
ADACMP utility, 43–82
  COMPRESS function, 60
    examples, 101
    output, 54
      compressed data records, 54
      rejected data records, 54
      storage requirements report, 55
    overview, 43
  DECOMPRESS function, 103
    examples, 108
    multiclient files, 107
    output, 57
      rejected data records, 58
    overview, 44
  input requirements
    data structure, 45
    multiple-value field count, 45
    periodic group count, 47
    variable-length field size, 50
  JCL requirements and examples, 108
    BS2000, 110
    OS/390 or z/OS, 113
    VM/ESA, 116
    VSE/ESA, 117
  processing, 52
    data compression, 52

data verification, 52
restart considerations, 59
user exit 6, 59
user exits
collation, 109
compression, 108
ADACNV utility, 121–135
CONVERT function, 123
functional overview, 121
JCL requirements and examples, 129
BS2000, 129
OS/390 or z/OS, 132
VM/ESA or z/VM, 133
VSE/ESA, 134
REVERT function, 126
ADADBS utility, 137–167
ADD dataset function, 139
ALLOCATE function, 141
CHANGE function, 143
checking syntax, 138
CVOLSER function, 145
DEALLOCATE function, 146
DECREASE function, 148
DELCP function, 149
DELETE function, 151
DSREUSE function, 153
ENCODEF function, 155
functional overview, 137
INCREASE function, 156
procedure
BS2000, 163
general, 157
OS/390 or z/OS, 158
VM/ESA or z/VM, 161
VSE/ESA, 159
ISNREUSE function, 164
JCL requirements and examples
BS2000, 214
OS/390 or z/OS, 215
VM/ESA or z/VM, 216
VSE/ESA, 217
MODFCB function, 165
NEWFIELD function, 168
ONLINVERT function, 171
ONLREORFASSO function, 173
ONLREORFDATA function, 175
ONLREORFILE function, 178
OPERCOM function, 181
PRIORITY function, 198
RECOVER function, 199
REFRESH function, 200
REFRESHSTATS function, 201
RELEASE function, 203
RENAME function, 205
RENUMBER function, 206
RESETDIB function, 207
TRANSACTIONS function, 209
UNCOUPLE function, 212
ADADCK utility, 219–226
DSCHECK function, 220
JCL requirements and examples
BS2000, 222
OS/390 or z/OS, 223
VM/ESA or z/VM, 225
VSE/ESA, 226
ADADEF utility, 227–250
DEFINE function, 228
functional overview, 227–250
checkpoint file, 227–250
database components, 227–250
JCL requirements and examples
BS2000, 242
OS/390 or z/OS, 245
VM/ESA or z/VM, 247
VSE/ESA, 249
MODIFY function, 237
NEWWORK function, 240
ADAEND operator command, using utility, 183
ADAFRM utility, 251–263
all functions, 252, 253
format new RABNs, 251
formatting database components, 251
functional overview, 251
JCL requirements and examples

BS2000, 257
OS/390 or z/OS, 259
VM/ESA or z/VM, 261
VSE/ESA, 262
reset dataset blocks/cylinders to zeros, 251
ADAICK utility, 265–290
ACCHECK function, 267
ASSOPRINT function, 268
BATCH function, 269
DATAPRINT function, 270
DSCHECK function, 271
DUMP function, 272
examples, 285
FCBPRINT function, 273
FDTPRINT function, 274
functional overview, 265–290
GCBPRINT function, 275
ICHECK function, 276
INT function, 277
JCL requirements and examples
BS2000, 286
OS/390 or z/OS, 288
VM/ESA or z/VM, 289
VSE/ESA, 290
NIPRINT function, 278
NOBATCH function, 279
NODUMP function, 280
NOINT function, 281
PPTPRINT function, 282
UIPRINT function, 284
user exits, collation, 286
ADAINV utility, 291–312
COUPLE function, 292
examples, 302
functional overview, 291
INVERT function, 298
JCL requirements and examples
BS2000, 304
OS/390 or z/OS, 307
VM/ESA or z/VM, 309
VSE/ESA, 311
space allocation during execution, 302
user exits, collation, 213, 303
ADALOD utility, 313–371
JCL requirements and examples
BS2000, 359
OS/390 or z/OS, 363
VM/ESAor z/VM, 366
VSE/ESA, 369
LOAD function, 314
Associator updating by, 337
examples, 332–371
input data for, 334
space allocation for file, 334
space/statistics report, 356
storage requirements and use, 354
Temp dataset requirements, 355
UPDATE function, 341
Associator updating with, 352
descriptor information generation, 352
examples, 348
input requirements, 351
mass updates to expanded files, 353
space allocation, 352
user exits, collation, 358
ADAM
load files with, using utility, 318
retrieval statistics, using utility, 373–383
ADAMER utility, 373–383
examples, 377
JCL requirements and examples
BS2000, 379
OS/390 or z/OS, 380
VM/ESA or z/VM, 381
VSE/ESA, 382
output report, 378
syntax, 374
Address converter
allocate an extent, using utility, 141
check against Data Storage, using utility, 12
check index against, using utility, 276
deallocate an extent, using utility, 146
space allocation, using utility, 335, 352

validate for specific files, using utility, 265–290
ALOCKF operator command, using utility, 183
Alphanumeric fields, no conversion option (NV), 76
Associator
add dataset to, using utility, 139
check physical structure of, using utility, 265
coupling lists, creating, 296
decrease size of dataset, using utility, 148
format, using utility, 252
increase size of dataset, using utility, 156
print/dump block(s), using utility, 268
reset blocks/cylinders to zeros, using utility, 253
updating
using utility, 352
using utlity, 337
ATM, loading system files for, 316
Attached buffers, command to display usage, 189

C

CANCEL operator command, using utility, 184
Checkpoint file
define, using utility, 228
define for a new database, using utility, 227
Checkpoints, delete, using utility, 149
Collation descriptor, define
using ADACMP, 83
using ADAINV, 298
Command log
close/switch dual, using utility, 191
format, using utility, 253
start logging, using utility, 192
stop logging, using utility, 192
Command queue, command to display usage, 189
Command queue element
display, using utility, 188
display posted, using utility, 186
Convert database
to higher version, 123
to lower version, 126
Coparameters, specifying, 8
CT, ADARUN parameter, command to override setting, 186
CT operator command, using utility, 186

D

Data compression
ADACMP utility, 43, 52
fields with NC option, 79
Data decompression
ADACMP utility, 44, 103
fields with NC option, 79
Data definition
COLDE statement
of ADACMP COMPRESS, 83
of ADAINV INVERT, 299
field options
DE – descriptor, 70
FI – fixed storage, 71
LA – long alphanumeric, 72
MU – multiple-value, 73
NC – null not counted, 79
compressing/decompressing, 79
NN – not null, 81
NU – null value suppression, 75
NV – no conversion, 76
overview, 70
PE – periodic group, 76
UQ – unique descriptor, 77
XI – exclude PE instance from UQ, 77
FIELD statement, of ADAINV INVERT, 299
FNDEF statement, of ADACMP COMPRESS, 66
HYPDE statement
of ADACMP COMPRESS, 85
of ADAINV INVERT, 299
PHONDE statement

of ADACMP COMPRESS, 89
of ADAINV INVERT, 299
SUBDE statement
of ADACMP COMPRESS, 90
of ADAINV INVERT, 299
SUBFN statement, 93
SUPDE statement
of ADACMP COMPRESS, 94
of ADAINV INVERT, 299
SUPFN statement, 99
syntax, using ADACMP utility, 66
Data integrity block
display entries, using utility, 187
reset entries in, using utility, 207
Data Storage
add dataset to, using utility, 139
allocate an extent, using utility, 141
check for a specified file, using utility, 219–226
check the address converter against, using utility, 12
deallocate an extent, using utility, 146
decrease size of dataset, using utility, 148
format, using utility, 252
increase size of dataset, using utility, 156
print/dump block(s), using utility, 270
print/dump record, using utility, 271
reset blocks/cylinders to zeros, using utility, 253
reuse blocks, using utility, 153
space allocation, using utility, 336
Database
change name assigned to, using utility, 205
component datasets, format, using utility, 251–263
define, using utility, 228
define a new, using utility, 227–250
delete file from, using utility, 151
Datasets
adding/updating, using utility, 313–371
format, using utility, 251–263
intermediate coupling storage, calculate using utility, 295
DAUQ operator command, using utility, 186
DCQ operator command, using utility, 186
DDIB operator command, using utility, 187
DDSF operator command, using utility, 187
Descriptor
collation, 83
define, using ADAINV utility, 291–312
field option (DE), 70
hyperdescriptor, 85
phonetic, 89
release from descriptor status, using utility, 203
specify for file coupling, using utility, 292
subdescriptor, 90
superdescriptor, 94
unique, 77
DFILES operator command, using utility, 187
DFILUSE operator command, using utility, 187
DHQ operator command, using utility, 188
DHQA operator command, using utility, 188
Disk volume, print Adabas extents located on, using utility, 145
DLOCKF operator command, using utility, 188
DNC operator command, using utility, 188
DNH operator command, using utility, 188
DNU operator command, using utility, 188
DONLSTAT operator command, using utility, 189
DPARM operator command, using utility, 189
DRES operator command, using utility, 189
DSTAT operator command, using utility, 190
DTH operator command, using utility, 190
Dump
suppress, using utility, 280
suspend suppression of, using utility, 272
terminate online status, using utility, 194
DUQ operator command, using utility, 190
DUQA operator command, using utility, 190
DUQE operator command, utility, 190

DUUQE operator command, using utility, 190

E

Encodings, change, using utility, 237
Expanded file chain, checking uniqueness of descriptor within, 77
Expanded files
  load anchor file, 320
  loading, 337
  mass updates to, 353

F

FEOFCL operator command, using utility, 191
FEOFPL operator command, using utility, 191
Field definition table
  add a field to, using utility, 168
  print/dump, using utility, 274
Fields
  add, using utility, 168
  change standard length of, using utility, 143
  subfield, 93
  superfield, 99
File control block, dump/print, using utility, 273
File coupling
  lists
    create using utility, 296
    space for, calculate using utility, 297
  temporary space for, calculate using utility, 295
  uncouple, using utility, 212
  using utility, 291–312
File encoding, modify, 155
File extents
  allocate, using utility, 141
  deallocate, using utility, 146
  print on given disk volume, using utility, 145
Files
  Adabas, create using utility, 60
Adabas sequential
  BS2000
    concatenate input, 391
    control statement read procedure, 393
    record formats, 389
    tape release option, 393
  determining names and blocks sizes, 385
  list by utility, 385
  OS/390 or z/OS, record formats, 394
  VM/ESA or z/VM, record formats, 395
  VSE/ESA
    concatenation of input, 397
    record formats, 396
allocate space for, 334
change file number of, using utility, 206
change name assigned to, using utility, 205
decompressing, multiclient, 107
delete, using utility, 151
display
  locked, using utility, 188
  total commands processed for select, using utility, 187
  user types for select, using utility, 187
load, using utility, 314
load with ADAM option, 318
lock
  at all security levels, using utility, 191
  for all non-utility use, using utility, 191
  for all users except EXU/EXF, using utility, 192
  in advance, using utility, 183
modify parameters of, using utility, 165
remove advance lock
  on all files, 194
  on specified file, 194
reset to empty status, using utility, 200
stop users of specified, using utility, 195
system, Checkpoint, define for a new database, 227
types, 316
unlock specified
  for utility use, using utility, 197

using utility, 197
Fixed storage (FI), use in ADACMP, 71
Format buffer
start logging, using utility, 192
stop logging, using utility, 192
Format pool, command to display usage, 189

G

General control block, print/dump, using utility, 275

H

HALT operator command, using utility, 191
Hold queue
command to display usage, 189
display count of ISNs, using utility, 188
Hold queue element, display, using utility, 188
Hyperdescriptor, define
using ADACMP, 85
using ADAINV, 298

I

I/O activity
start logging, using utility, 192
stop logging, using utility, 193
Index
check against address converter, using utility, 276
space allocation
by ADAINV, 302
for coupling lists, using utility, 298
using utility, 334
validate for specific files, using utility, 265–290
Invert, start online process, using utility, 171
ISN buffer
start logging, using utility, 192
stop logging, using utility, 192
ISNs
format for specifying, 349
set to resuse, using utility, 164

L

Lock, file in advance, command to set, 183
LOCKF operator command, using utility, 191
LOCKU operator command, using utility, 191
LOCKX operator command, using utility, 192
LOGGING, operator command, using utility, 192
LOGxx, operator command, using utility, 192
Long alphanumeric (LA), field option, 72

M

Multiclient files
decompressing, 107
loading, 340
owner ID
assign a new, using utility, 324
specify length of, using utility, 325
Multiple-value fields, data definition option (MU), 73

N

NOLOGGING, operator command, using utility, 192
NOLOGxx, operator command, using utility, 192
Normal index
allocate an extent, using utility, 141
deallocate an extent, using utility, 146
print/dump, using utility, 278
Nucleus, display current operating status, using utility, 190

Null value
indicator, specified in record buffer, 80
not allowed (NN), field option, 81
not counted (NC), field option, 79
SQL support, 78
suppression (NU), field option, 75

O

Online invert, start, using utility, 171
Online process
display status of, using utility, 189
resume a suspended process, using utility, 193
stop cleanly, using utility, 193
suspend, using utility, 193
Online reorder
Associator, 173
Data Storage, 175
file, 178
ONLRESUME, operator command, using utility, 193
ONLSTOP, operator command, using utility, 193
ONLSUSPEND, operator command, using utility, 193
Operator commands, ADADBS OPERCOM function, 181

P

Parallel participant table, print/dump, using utility, 282
Parameters, positional values, specifying, 8
Periodic groups, data definition option (PE), 76
Phonetic descriptor, define
using ADACMP, 89
using ADAINV, 298
Printout
formatted
cancel suppression of, using utility, 277
suppress, using utility, 281
set width to 132 characters, using utility, 269
set width to 80 characters, using utility, 279
Procedures, for defining Adabas libraries, VSE/ESA, 401
Protection log
close/switch dual, using utility, 191
format, using utility, 253

Q

Quiesce database, ADADBS function, 209

R

RALOCKF operator command, using utility, 194
RALOCKFA operator command, using utility, 194
RDUMPST operator command, using utility, 194
READONLY, operator command, using utility, 194
Read-only status, switch on/off, using utility, 184, 194
Record buffer
null value indicator value, 80
start logging, using utility, 192
stop logging, using utility, 193
Records, add/delete, using utility, 341
Recovery log, format, using utility, 253
Redo pool, command to display usage, 189
Reorder Associator, start online process, using utility, 173
Reorder Data Storage, start online process, using utility, 175
Reorder file, start online process, using utility, 178
Resources
display current usage, using utility, 189
statistics, command to display, 189

Resume normal processing, ADADBS function, 209
Revert database, to lower version, 126
REVIEW, operator command, using utility, 195

S

Search buffer
  start logging, using utility, 192
  stop logging, using utility, 193
Security pool, command to display usage, 189
Session
  cancel immediately, using utility, 184
  display current parameters, using utility, 189
  reset statistical values for, using utility, 201
  stop, using utility, 191
  terminate normally, using utility, 183
Sort, format, using utility, 253
Space
  estimation report (ADACMP), 55
  for file coupling lists, calculate using utility, 297
  recover, using utility, 199
  temporary, for file coupling, calculate using utility, 295
SQL, null representation support, 78
STOPF, operator command, using utility, 195
STOPI, operator command, using utility, 195
STOPU, operator command, using utility, 195
Storage, fixed (FI), 71
Subdescriptor, define
  using ADACMP, 90
  using ADAINV, 298
Subfield, define, using utility, 93
Subparameters, specifying, 8
Superdescriptor, define
  using ADACMP, 94
  using ADAINV, 298
Superfield, define, using utility, 99
Suspend normal processing, ADADBS function, 209
SYNCC, operator command, using utility, 196

T

Table of ISNs pool, command to display usage, 189
Table of sequential commands pool, command to display usage, 189
Temp
  ADALOD requirements for, 355
  format, using utility, 253
  space allocation, using utility, 336
Threads, display status, using utility, 190
Timeout control
  interregion communication limit, command to override setting, 186
  non-activity limit
    set for access-only users, using utility, 196
    set for ET logic users, using utility, 196
    set for exclusive control users, using utility, 196
TNAA, operator command, using utility, 196
TNAE, operator command, using utility, 196
TNAX, operator command, using utility, 196
Transaction, set time limit for ET logic users, using utility, 196–197
Transaction ID (XID) pool, command to display usage, 189
Transaction processing, suspend/resume, 209
TT, operator command, using utility, 196

U

Unique descriptor
  define, using ADALOD, 331
  exclude PE instance, 77
  use in ADACMP, 77
Unique descriptor pool, command to display usage, 189
Universal encoding support (UES)
  no conversion field option (NV), 76

table of encodings, 403–410
coexistent, 407
interoperable, 404
UNLOCKF, operator command, using utility, 197
UNLOCKU, operator command, using utility, 197
UNLOCKX, operator command, using utility, 197
Upper index
allocate an extent, using utility, 141
deallocate an extent, using utility, 146
print/dump, using utility, 284
User
change priority, using utility, 198
display count of, using utility, 188
ET logic, resynchronize all, using utility, 196
set non-activity time limit, using utility, 196
stop those timed out, using utility, 195
stop those using a specified file, using utility, 195
stop those with a specified job name, using utility, 196
stop user with specified ID, using utility, 196
User data
start logging, using utility, 192
stop logging, using utility, 193
User exits
6, ADACMP user processing, 59
ADACDU, 29
hyperdescriptor, 85
User queue, command to display usage, 189
User queue element
display, using utility, 186
display all, using utility, 190
display for specified user, using utility, 190
display up to 5, using operator command, 190
display utility, using utility, 190
remove stopped, using utility, 195
User queue file list pool, command to display usage, 189
Utility control statement
parameter values, 6
default, 9
value, 7
value list, 8
value range, 8
rules, 6
syntax, 2
mutually exclusive parameters, 4
parameter list, 2
repeating parameters and values, 5
required and optional parameters, 3
subparameters, 3
Utility-only status, switch on/off, using utility, 197
UTIONLY, operator command, using utility, 197

V

Value buffer
start logging, using utility, 192
stop logging, using utility, 193
VSE/ESA, procedures for examples, 401

W

Wide-character fields, no conversion option (NV), 76
Work
define, for an existing database, using utility, 227
format, using utility, 253
reset blocks/cylinders to zeros, using utility, 253
Work file, define, using utility, 240
Work pool, command to display usage, 189

X

XID pool, command to display usage, 189